Популярные лекции
по математике

Б. А. ТРАХТЕНБРОТ

АЛГОРИТМЫ И МАШИННОЕ РЕШЕНИЕ ЗАДАЧ

*

ГОСУДАРСТВЕННОЕ ИЗДАТЕЛЬСТВО
ТЕХНИКО-ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА · 1957

ПОПУЛЯРНЫЕ ЛЕКЦИИ ПО МАТЕМАТИКЕ
ВЫПУСК 26

Б. А. ТРАХТЕНБРОТ

АЛГОРИТМЫ И МАШИННОЕ РЕШЕНИЕ ЗАДАЧ

ГОСУДАРСТВЕННОЕ ИЗДАТЕЛЬСТВО
ТЕХНИКО-ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1957

11-2-1

АННОТАЦИЯ

Книга Б. А. Трахтенброта рассматривает в популярной форме основные вопросы теории алгоритмов и связь этой теории с современной машинной математикой. Автор подробно рассказывает об истории развития поиятия алгоритм, о принципе работы современных быстродействующих вычислительных машин, об основах программирования, о схеме машины Тьюринга, об алгоритмически неразрешимых проблемах.

Книга рассчитана на школьников старших классов, преподавателей, инженерно-технических работников и всех лиц, интересующихся перспективами применения новой вычислительной техники.

СОДЕРЖАНИЕ

Предисловие 4

Введение 5

§ 1. Числеиные алгоритмы 7
§ 2. Алгоритмы для решения логических задач 12
§ 3. Проблема слов 23
§ 4. Вычислительиая машина с автоматическим управлением . 37
§ 5. Программа (машииный алгоритм) 44
§ 6. Необходимость уточнения понятия алгоритма 52
§ 7. Машина Тьюринга 60
§ 8. Реализация алгоритма в машине Тьюринга 67
§ 9. Основная гипотеза теории алгоритмов 79
§ 10. Универсальная машина Тьюринга 82
§ 11. Алгоритмически неразрешимые проблемы 89

Заключительные замечания 94

ПРЕДИСЛОВИЕ

Настоящая книга, являющаяся элементарным введением в теорию алгоритмов, посвящена разъяснению одного из самых основных понятий математики — понятия алгоритма; в ней рассматривается круг вопросов, лежащих на грани между математической логикой и теорией автоматических вычислительных машин.

В основу книги легли популярные лекции и обзорные доклады, с которыми автор выступал в г. Пензе, начиная с 1951 года, перед разными аудиториями, а также одноименная статья, написанная им для журнала «Математика в школе» (№№ 4—5, 1956 г.).

Тем, кто пожелает более основательно ознакомиться с этими вопросами, можно рекомендовать следующую литературу:

1. А. А. Марков, Теория алгорифмов, Труды матем. ин-та им. Стеклова, т. XLII, Изд-во Академии наук СССР, Москва, 1954 г.

2. Р. Петер, Рекурсивные функции, Москва, 1954 г.

3. А. И. Китов, Электронные вычислительные машины, Изд-во Советское радио, Москва, 1956 г.

4. С. Клини, Введение в метаматематику, ИЛ, Москва, 1956 г.

ВВЕДЕНИЕ

В послевоенные годы получили большое развитие автоматические быстродействующие вычислительные машины, которые теперь применяются для решения самых разнообразных математических и логических задач. Характерная особенность этих машин, отличающая их от прежних вычислительных машин, заключается в том, что при выполнении своих функций, начиная с того момента, как в них введены начальные данные и программа, они работают без всякого вмешательства человека вплоть до выдачи окончательного результата. Производительность современных электронных автоматических машин огромна: они способны выполнить в одну секунду до 20 000 арифметических операций, что по крайней мере в 10 раз больше того, что может сделать за смену квалифицированный вычислитель, работающий на хорошем клавишном арифмометре *). Область применения автоматических машин продолжает расти: машины решают сложные системы уравнений, переводят с одного языка на другой, играют в шахматы и т. д. Очень велики перспективы применения на производстве автоматических машин, осуществляющих управление всем технологическим процессом в масштабе крупного завода. Кроме того, возможность быстрой и надежной математической обработки, а также анализа экспериментальных данных создает предпосылки для появления новых, ранее недоступных методов исследования во многих областях науки.

Теперь уже все признают, что автоматические вычислительные машины представляют собою мощные орудия умственного труда, способные не только облегчить умственный труд человека, но и полностью освободить человека от некоторых видов большой и напряженной умственной работы.

*) С точки зрения производства операций.

Вместе с тем достигнутые успехи могут породить и действительно порождают много неоправданных иллюзий и чисто фантастических прогнозов по поводу всемогущества машин. Особо следует указать на рекламную шумиху, поднятую в части зарубежной печати о «гигантском электронном мозге», об автоматах, способных решать любые задачи и заменить творческий труд ученого.

В связи с указанными обстоятельствами приобретает большую актуальность и остроту вопрос о том, какие виды умственной работы могут выполнять автоматические вычислительные машины. Этот вопрос с определенной точки зрения рассматривается и решается в современной теории алгоритмов, являющейся важной ветвью математической логики. Для математической логики характерно исследование сущности таких понятий, как «вычислительный процесс», «математическое доказательство», «алгоритм» и т. д. Еще за несколько лет до создания современных электронных автоматических вычислительных машин в математической логике были выработаны точные понятия «алгоритм» и общая схема автоматической вычислительной машины (машина Тьюринга), а также выяснена тесная связь между алгоритмами и машинами. Это позволило установить ряд важных теорем, раскрывающих сущность процессов, реализуемых в автоматических машинах; в частности, было строго доказано существование таких проблем, для которых невозможно машинное решение. Рассмотрению связи между алгоритмами и машинами и посвящена предлагаемая книга.

В §§ 1—3 на ряде примеров разъясняется, что такое алгоритм, и строятся алгоритмы для решения некоторых классов математических и логических задач.

В §§ 4—5 изложены принципы устройства электронных вычислительных машин и составления программ, т. е. алгоритмов, приспособленных для реализации их в машинах.

Параграфы 6—11 посвящены ряду важных фактов теории алгоритмов, причем в качестве основного понятия теории взято понятие машины Тьюринга.

Чисто техническая громоздкость многих доказательств не позволяет в такой небольшой книжке давать их полное изложение. Поэтому имеются некоторые отступления от строгости и полноты изложения, которые, однако, как нам кажется, не мешают, а, наоборот, благоприятствуют лучшему уяснению существа дела. Для общности картины кое-что дано в обзорном порядке (§ 6).

Сделаем еще одно замечание. Современные машины с автоматическим управлением называют электронными, так как их важнейшие органы сконструированы из электронных ламп. Применение электронной техники обеспечивает большую экономию во времени на реализацию отдельных операций, совершаемых машиной. Однако основная черта этих машин — автоматическое управление процессами, происходящими в них, — не возникает за счет применения именно электронной техники. В принципе, электронные лампы можно было бы заменить даже механическими устройствами, т. е. можно было бы создать механическую вычислительную машину с автоматическим управлением, которая способна решать те же задачи, что и электронная (правда, значительно медленнее). Таким образом, возникновение вычислительных машин нового типа нельзя рассматривать как результат развития только электронной техники. Более того, первое описание общей схемы автоматической вычислительной машины (машина Тьюринга, см. № 7) было дано в теории алгоритмов еще в 1936 году как описание механического устройства. Первые фактически сконструированные машины (1940 г.) были электромеханическими.

В предлагаемой книге при описании устройства машин мы не будем останавливаться на технических деталях, основное внимание будет уделяться рассмотрению принципов взаимодействия отдельных органов машины. Такой подход соответствует главной цели книги, заключающейся в раскрытии математических и логических возможностей машин, а не в показе технической стороны дела.

§ 1. ЧИСЛЕННЫЕ АЛГОРИТМЫ

Понятие алгоритма принадлежит к числу основных понятий математики. Под *алгоритмом* понимают точное предписание о выполнении в определенном порядке некоторой системы операций для решения всех задач некоторого данного типа.

Разумеется, высказанная фраза не является точным математическим определением понятия *алгоритм*, она скорее толкует значение слова «*алгоритм*», поясняя его смысл. Тем не менее эта формулировка близка и понятна каждому математику; она отражает то понятие алгоритма, которое стихийно складывалось и применялось в математике с древнейших времен.

Простейшими алгоритмами являются правила, по которым выполняется то или другое из четырех арифметических действий по десятичной системе счисления (сам термин *алгоритм* происходит от имени средневекового узбекского математика Аль-Хорезми, который еще в IX веке дал такие правила). Так, например, действие сложения двух многозначных чисел разлагается на цепочку элементарных операций, при осуществлении каждой из которых вычислитель обозревает лишь 2 соответствующие цифры слагаемых (одна из которых может оказаться снабженной пометкой, напоминающей о переносе единицы). Эти операции бывают двух типов: 1) запись соответствующей цифры суммы, 2) пометка о переносе над соседней слева цифрой; при этом правило предписывает надлежащий порядок выполнения этих операций (справа налево). Формальный характер этих элементарных операций заключается в том, что они могут быть выполнены автоматически по раз навсегда заданной таблице сложения цифр, при полном отвлечении от их содержательного смысла.

Аналогично обстоит дело с остальными тремя арифметическими действиями, а также с действием извлечения корня квадратного и другими. Формальный характер соответствующих предписаний (алгоритмов) по-видимому не вызывает никаких сомнений (для школьников это особенно заметно в правиле извлечения корня).

В качестве дальнейшего примера рассмотрим *алгоритм Евклида*, решающий все задачи следующего типа:

Для данных двух натуральных чисел a, b найти их общий наибольший делитель.

Очевидно, различных задач такого типа существует столько, сколько различных пар чисел a, b.

Как известно, решение любой из этих задач можно получить путем построения убывающей последовательности чисел, из которых первое является бо́льшим из двух данных, второе — меньшим, третье получается как остаток от деления первого на второе, четвертое — как остаток от деления второго на третье и так далее, пока не будет совершено деление без остатка. Делитель в этом последнем делении и будет искомым результатом.

Поскольку деление сводится к повторному вычитанию, предписание, пригодное для решения любой из этих задач, можно было бы задать в виде следующей последовательности указаний:

Указание 1. Обозревай 2 числа: a, b. Переходи к следующему указанию.

Указание 2. Сравни обозреваемые числа ($a = b$, или $a < b$, или $a > b$); переходи к следующему указанию.

Указание 3. Если обозреваемые числа равны, то каждое из них дает искомый результат. Процесс вычисления остановить. Если нет, переходи к следующему указанию.

Указание 4. Если первое обозреваемое число меньше второго обозреваемого числа, переставь их местами и продолжай обозревать их. Переходи к следующему указанию.

Указание 5. Вычитай второе из обозреваемых чисел из первого и обозревай два числа: вычитаемое и остаток. Переходи к указанию 2.

Итак, после того, как все 5 указаний выполнены, надо опять возвратиться ко второму, потом дальше к третьему, четвертому, пятому и опять ко второму, третьему и т. д., пока не получатся равные числа, т. е. пока не окажется выполненным условие, содержащееся в третьем указании; тогда процесс прекращается.

Правда, в математике алгоритмы не всегда формулируются в таком педантично формальном виде; однако возможность такого формального задания любого из известных алгоритмов, по-видимому, ни у кого сомнения не вызывает.

В приведенном описании алгоритма Евклида в качестве элементарных операций, на которые расчленяется процесс решения задачи, фигурирует вычитание двух чисел, сравнение двух чисел и перестановка двух чисел местами, но легко понять, что это расчленение может быть продвинуто гораздо дальше. Так, например, указание 5 о вычитании двух обозреваемых чисел может быть само развернуто в систему указаний, описывающих алгоритмы вычитания двух чисел. Однако в силу большой простоты и привычности правил арифметических действий в подобных случаях дальнейшая детализация алгоритма не проводится.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к четырем арифметическим действиям, называются *численными алгоритмами*. Они играют важную роль в самых разнообразных областях как элементарной, так и высшей математики и задаются обычно в виде словесных предписаний или же разного рода формул и схем. Так,

например, алгоритм решения системы двух уравнений первой степени с двумя неизвестными:

$$a_1x + b_1y = c_1,$$
$$a_2x + b_2y = c_2$$

дается формулами

$$x = \frac{c_1b_2 - c_2b_1}{a_1b_2 - a_2b_1}; \qquad y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1},$$

в которых полностью выражен как состав действий, так и порядок их выполнения. В приведенных формулах предусмотрена одна и та же цепочка действий для всех задач данного типа (т. е. при любых коэффициентах a_1, b_1, c_1, a_2, b_2, c_2). Интересно, однако, отметить, что, вообще говоря, число операций, предписываемых алгоритмом, заранее не бывает известным; оно зависит от конкретного выбора условий задачи и выясняется лишь в процессе самого решения.

В частности, так обстоит дело и в случае алгоритма Евклида, где число вычитаний, могущих понадобиться, зависит от выбора той или иной пары чисел a, b.

Широкое распространение численных алгоритмов обусловливается тем, что к четырем арифметическим действиям можно свести очень многие другие операции. Правда, такое сведение обычно не является исчерпывающе точным, но оно может быть осуществлено с любой наперед заданной точностью. Все это можно было бы иллюстрировать уже на примере алгоритма извлечения корня квадратного, который позволяет находить корень приближенно, но с любой наперед заданной точностью, при помощи последовательности делений, умножений и вычитаний. В специальной ветви современной математики (*численный анализ*) разрабатываются аналогичные приемы сведения к арифметическим действиям и более сложных операций, как интегрирование, дифференцирование и т. д.

В математике серия *задач определенного типа считается решенной, когда для ее решения установлен алгоритм.* Нахождение таких алгоритмов является естественной целью математики. Так, например, в алгебре установлены алгоритмы, которые по заданным коэффициентам алгебраического уравнения позволяют совершенно автоматически определить, сколько различных корней имеет данное уравнение (и какой кратности), и вычислить эти корни с любой наперед заданной точностью.

Если же математик не обладает алгоритмом для решения всех задач данного типа, то хотя порою и удается решать некоторые единичные задачи этого типа, но в отдельных случаях приходится придумывать специальную процедуру, непригодную уже для большинства других случаев.

Приведем пример такого класса однотипных задач, для решения которых современная математика не располагает алгоритмом.

Рассмотрим всевозможные диафантовы уравнения, т. е. уравнения вида

$$P = 0,$$

где P является многочленом с целочисленными коэффициентами. Такими будут, например, уравнения

$$x^2 + y^2 - z^2 = 0,$$
$$6x^{18} - x + 3 = 0,$$

из которых первое — с двумя неизвестными, а второе с одним неизвестным (вообще же рассматриваются уравнения с любым числом неизвестных). Уравнение может иметь целочисленное решение, а может и не иметь такового. Так, первое из приведенных уравнений имеет целочисленное решение

$$x = 3,\ y = 4,\ z = 5;$$

второе же уравнение не имеет целочисленного решения, ибо для любого целого x легко устанавливается неравенство

$$6x^{18} > x - 3.$$

В 1901 году на международном математическом конгрессе в Париже знаменитый немецкий математик Давид Гильберт огласил список 20 трудных проблем, на важность решения которых он обращал внимание математической общественности. Среди них была и следующая проблема (10-я проблема Гильберта): *требуется выработать алгоритм, позволяющий для любого диофантова уравнения выяснить: имеет ли оно целочисленное решение.*

Для частного случая диофантова уравнения с одним неизвестным такой алгоритм уже давно известен. Именно, установлено, что если уравнение

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0$$

с целочисленными коэффициентами имеет целый корень x_0, то обязательно a_0 делится на x_0. В соответствии с этим можно предложить такой алгоритм:

1) найти все делители числа a_0 (их конечное число);

2) подставлять поочередно найденные делители в левую часть уравнения и вычислять численное значение ее;

3) если при подстановке некоторого делителя левая часть уравнения приняла значение 0, то данный делитель является корнем уравнения; если же ни для какого делителя левая часть уравнения не обращается в 0, то уравнение не имеет целочисленных корней.

Проблема Гильберта привлекала и продолжает привлекать внимание многих выдающихся математиков, но тем не менее в общем случае, когда дано уравнение с двумя или многими неизвестными, требуемый алгоритм неизвестен и по сей день. Более того, теперь уже кажется весьма правдоподобным, что такой алгоритм никогда не будет найден и в будущем. Однако точный смысл этого пессимистического на первый взгляд прогноза станет ясным читателю лишь позднее из дальнейшего изложения.

Уже в рассмотренных до сих пор примерах довольно отчетливо выступают следующие черты численных алгоритмов, присущие и любому другому алгоритму, а именно:

Детерминированность алгоритма. Требуется, чтобы метод вычисления можно было сообщить другому лицу в виде конечного числа указаний, как действовать на отдельных стадиях вычисления. При этом вычисления согласно этим указаниям не зависят от произвола вычисляющего лица и представляют собою детерминированный процесс, который может быть в любое время повторен и выполнен с тем же успехом и другим лицом.

Массовость алгоритма. Алгоритм—это единое предписание, определяющее вычислительный процесс, который может начинаться от различных исходных данных и ведет во всех случаях к соответствующему результату. Иными словами—алгоритм решает не одну лишь индивидуальную задачу, а некоторую серию однотипных задач.

§ 2. АЛГОРИТМЫ ДЛЯ РЕШЕНИЯ ЛОГИЧЕСКИХ ЗАДАЧ

Задачи, рассмотренные в предыдущем параграфе, были взяты из арифметики, алгебры, теории чисел; они являются достаточно типичными для проблематики этих областей математики и носят, так сказать, традиционный математиче-

ский характер. Мы разберем теперь два класса задач, которые имеют несколько иной характер; их можно было бы скорее назвать логическими, чем математическими, хотя и весьма затруднительно предложить достаточно четкий критерий для различения подобных *логических* задач от обычных *математических* задач. Не вдаваясь в дальнейшее обсуждение терминологии, заметим лишь, что в обоих случаях речь идет опять-таки о нахождении алгоритма, позволяющего единым методом решать любую задачу из рассматриваемого класса однотипных задач; однако в рассматриваемых случаях эти алгоритмы уже не являются численными.

1. Игра «15». Квадратная доска разбита на 16 равных клеток, причем в любых 15 из них можно произвольным образом разместить (по одной в клетке) 15 фишек, занумерованных числами от 1 до 15, образуя таким образом некоторую *позицию*. Мы будем называть две клетки *соседними*, если их границы содержат общий отрезок. Две позиции считаются *смежными*, если можно преобразовать одну из них в другую в результате одного *хода*, заключающегося в передвижении в пустую клетку фишки с какой-либо соседней клетки.

Число всевозможных позиций конечно и равно

$$16! = 20\,922\,789\,888\,000,$$

причем с каждой позиции возможно лишь конечное число ходов (2, 3 или 4).

Возникает следующая серия однотипных задач:

Для любых двух позиций A и B выяснить, переводимы ли они одна в другую посредством конечной последовательности ходов.

Если ответ на поставленный вопрос утвердительный, то указанная последовательность ходов порождает цепочку позиций:

$$A \longleftrightarrow A_1 \longleftrightarrow A_2 \longleftrightarrow \ldots \longleftrightarrow A_n \longleftrightarrow B,$$

ведущую от A к B, где $\longleftrightarrow$ обозначает ход, переводящий позицию в смежную ей позицию. Можно считать, что в этой цепочке нет повторяющихся позиций, ибо в противном случае весь участок цепочки между двумя повторяющимися позициями можно было бы удалить и тем самым получить более экономную цепочку, ведущую от A к B. Итак, если ответ утвердителен, то число потребных ходов

не более 16!—1. Исходя из этих соображений, можно теперь сформулировать для поставленной проблемы следующий разрешающий алгоритм, основанный на простой идее переборки всевозможных комбинаций из 1, 2, 3...16!—1 ходов. Составляется список позиций, в который включаются: позиция A, позиции, смежные с A, позиции, смежные с этими смежными, и т. д. 16!—1 раз. Если в этом списке встречается B, то ответ утвердительный, если нет — то ответ отрицательный.

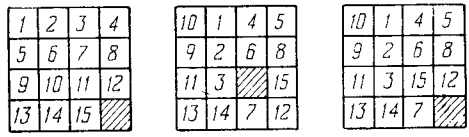
Хотя предложенный метод решения очень громоздок, все же на вопрос, обладаем ли мы точным предписанием, позволяющим единым методом в результате конечного числа шагов решать любую задачу рассматриваемого типа, мы должны дать утвердительный ответ. При этом мы, очевидно, имеем в виду потенциальную осуществимость (практическое осуществление зависит от средств, имеющихся в распоряжении вычислителя) описанного процесса, который является конечным, хотя и очень длинным.

При всей непривлекательности этого алгоритма его обнаружение нами представляет собою важное обстоятельство. Ведь для проблемы Гильберта о диофантовых уравнениях не удалось обнаружить никакого алгоритма! А между тем обнаружение алгоритма, пусть и громоздкого, может подать надежду и на его улучшение или создание более удобных алгоритмов. В частности, в нашем случае так оно и есть. Мы опишем теперь другой алгоритм, который является вполне приемлемым в смысле удобства, но для обоснования его потребуется одна теорема, которую мы ниже приведем без доказательства.

Назовем *транспозицией* перемену местами двух фишек на доске. Заметим, что всякие 2 позиции A и B переводимы друг в друга посредством передвижений и транспозиций; именно, если в этих позициях пустые клетки не совпадают, то сперва, при помощи одних лишь передвижений (не более 15) можно B перевести в позицию B' с той же пустой клеткой, что и в A. Далее перевод B' в A осуществляется уже при помощи одних лишь транспозиций (не более, чем 15 транспозициями). Справедливость нашего утверждения становится очевидной при разборе какого-либо примера. Пусть речь идет о позициях, изображенных на рис. 1.

Передвигаем 15-ю, а потом 12-ю фишки и получаем позицию B'. Далее помещаем фишку с № 1 на то же место,

что и в A посредством транспозиции (1, 10), затем производим транспозиции (2, 10), (3, 4), (4, 5), (5, 9), (6, 10), (7, 10), (9, 11), (10, 11), (11, 15). Фишки 8, 12, 13, 14, 15 находятся на соответствующих местах и поэтому в транспозициях не участвуют.



Позиция A Позиция B Позиция B'

Рис. 1.

Приведем теперь без доказательства нужную нам теорему.

Теорема. *Если позицию B можно перевести в позицию A при помощи передвижений и четного числа транспозиций, то B можно перевести в A и при помощи одних лишь передвижений. Если B можно перевести в A при помощи передвижений и нечетного числа транспозиций, то B непереводима в A одними лишь передвижениями.*

Теперь уже ясно, как строится алгоритм для поставленной задачи: указанным выше способом при помощи передвижений и транспозиций (всего — не более $15+15=30$ ходов) переводим позицию B в позицию A; если при этом было израсходовано четное число транспозиций, то ответ на поставленный вопрос положителен, в противном случае — отрицателен. Так, в нашем примере число транспозиций четно, а, значит, позиция B действительно переводима в позицию A.

2. Поиски пути в лабиринте. Греческая мифология повествует о легендарном герое Тезее, который отважился проникнуть в лабиринт для того, чтобы разыскать в нем чудовищного Минотавра и убить его. Ему помогла выбраться из лабиринта Ариадна, давшая Тезею клубок ниток, один конец которых держала она сама. По мере углубления Тезея в лабиринт клубок разматывался; наматывая потом нитки, Тезей благополучно вернулся к выходу.

Об этой древней легенде напомнила сравнительно недавно автоматическая игрушка «Мышь в лабиринте», созданная американским математиком и инженером Клодом Шэнноном. В одном месте специального лабиринта ставится

вещь, условно именуемая куском сала, в другом — «мышь». Мышь начинает блуждать по лабиринту, делая при этом петли, пока находит «сало». Если повторно запустить «мышь» с того же места, то она уже прямо бежит к «пище», не делая петель. Мы рассмотрим здесь некоторую родственную задачу поиска пути в лабиринте (вернее серию таких однотипных задач) и опишем алгоритм, в соответствии с которым должны проводиться поиски для достижения цели, указанной в задаче.

Мы представим себе лабиринт в виде конечной системы площадок, от которых расходятся коридоры, причем каждый коридор соединяет две площадки (такие площадки будем называть смежными), но не исключается существование таких площадок, из которых можно пройти только в один коридор (такие площадки будем называть тупиками). Геометрически лабиринт можно представить в виде системы точек A, B, C, ... (изображающих площадки) и совокупности отрезков AB, BC, ... (изображающих коридоры), соединяющих некоторые пары этих точек (рис. 2).

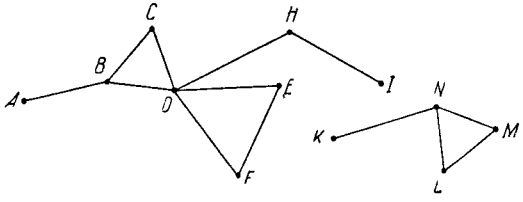


Рис. 2.

Будем говорить, что площадка Y достижима из площадки X, если существует путь, ведущий от X к Y через промежуточные коридоры и площадки. Точнее это означает, что либо X и Y — смежные площадки, либо существует последовательность площадок X_1, X_2, X_3, ..., X_n таких, что X и X_1, X_1 и X_2, X_2 и X_3, ... и т. д. и, наконец, X_n и Y смежны. Так, например, на рис. 2 площадка H достижима из тупика A посредством пути AB, BC, CD, DE, EF, FD, DH, в то время как площадка K не достижима из A. Вместе с тем, если Y вообще достижима из X, то она достижима и посредством *простого пути*, т. е. такого пути, в котором каждая площадка (а тем более и каждый коридор) проходится лишь один раз. В предыдущем при-

мере путь не был простым, но, срезав петлю DE, EF, FD, мы получаем простой путь AB, BC, CD, DH.

Предполагается, что Минотавр находится на одной из площадок лабиринта (обозначим ее через M), а Тезей, отправляясь на его поиски с площадки A, на которой его ждет Ариадна, должен решить следующую задачу: требуется выяснить, достижима ли M из A или нет*). Если достижима, то нужно добраться до нее по какому бы то ни было пути, но вернуться к Ариадне нужно уже по простому пути. Если же M не достижима, то вернуться к Ариадне.

Различных лабиринтов может быть бесчисленное множество, а взаимное расположение в данном лабиринте площадок A и M также можно варьировать. Поскольку Тезею заранее ничего не известно об устройстве данного лабиринта и о местонахождении в нем Минотавра, то решение поставленной задачи мыслится в виде общего метода поисков пригодного при любом лабиринте и при любом расположении в нем площадок A и M. Иными словами, решение мыслится в виде алгоритма, решающего любую из задач данного типа.

Для построения такого алгоритма мы рассмотрим один специальный метод поисков. На любой стадии процесса поиска в соответствии с этим методом приходится различать коридоры, еще ни разу не пройденные Тезеем (условно — зеленые), пройденные один раз (желтые), пройденные дважды (красные). Далее, находясь на какой-либо площадке, Тезей может попасть на одну из смежных площадок посредством одного из следующих двух ходов:

1. Разматывание нити. Прохождение от данной площадки по любому зеленому коридору до смежной площадки. (При этом нить Ариадны разматывается вдоль этого коридора, который после прохождения уже считается желтым.)

2. Наматывание нити. Возвращение от данной площадки по последнему пройденному желтому коридору до смежной площадки. При этом нить Ариадны, ранее размотанная вдоль этого коридора, обратно наматывается, а этот коридор уже объявляется красным.

Предполагается, что Тезей делает какие-то пометки, позволяющие ему впоследствии различать красные коридоры от зеленых; желтые различимы тем, что по ним протянута

*) Естественно считать, что $M \neq A$.

нить Ариадны. Выбор того или другого хода зависит от обстановки, наблюдаемой Тезеем на той площадке, на которой он находится в данный момент; эту обстановку можно охарактеризовать одним или несколькими из следующих признаков:

1. *Минотавр.* На данной площадке обнаружен Минотавр.
2. *Петля.* Через данную площадку уже протянута нить Ариадны; иными словами, от данной площадки расходятся по крайней мере два желтых коридора.
3. *Зеленая улица.* От данной площадки есть выход по крайней мере в один зеленый коридор.
4. *Ариадна.* На данной площадке находится Ариадна.
5. *Пятый случай.* Отсутствие всех предыдущих признаков.

Наш метод поиска может быть теперь задан следующей схемой:

Признак	Ход
1. *Минотавр*	Остановка
2. *Петля*	Наматывание нити
3. *Зеленая улица*	Разматывание нити
4. *Ариадна*	Остановка
5. *Пятый случай*	Наматывание нити.

Находясь на какой-нибудь площадке, Тезей делает очередной ход так: он проверяет по порядку номеров в соответствии с левым столбцом схемы, какой из перечисленных признаков имеет место; обнаружив первый такой признак, он (уже не проверяя остальные признаки) делает соответствующий ход (или остановку) из правого столбца. Такие ходы делаются до тех пор, пока не наступает остановка.

Пригодность предложенного метода непосредственно вытекает из следующих трех утверждений:

1. При любом взаимном расположении A и M в лабиринте после конечного числа ходов обязательно наступит остановка либо на площадке Минотавра, либо на площадке Ариадны.

2. Если остановка наступила на площадке Минотавра, то Минотавр достижим. Более того, в этом случае нить Ариадны оказывается протянутой по простому пути, ведущему от A до M; наматывая нить, Тезей может теперь вернуться по этому пути к Ариадне.

3. Если остановка наступила на площадке Ариадны, то Минотавр не достижим.

Прежде чем доказывать эти утверждения, покажем на двух примерах, как работает предложенный метод.

Пример 1. Пусть с площадки A лабиринта (рис. 2) начинается поиск Минотавра, который находится в F. Процесс поиска в соответствии с нашим методом удобно изобразить посредством схемы (в силу произвола при выборе зеленого коридора — это лишь одна из возможных схем), представленной на таблице 1.

Таблица 1

Порядковый номер хода	Каким признаком руководствуется Тезей	Ход	Пройденный коридор	Цвет коридора после его прохождения
1	*Зеленая улица*	Разматывание	AB	Желтый
2	» »	»	BC	»
3	» »	»	CD	»
4	» »	»	DH	»
5	» »	»	HJ	»
6	*Пятый случай*	Наматывание	JH	Красный
7	» »	»	HD	»
8	*Зеленая улица*	Разматывание	DB	Желтый
9	*Петля*	Наматывание	BD	Красный
10	*Зеленая улица*	Разматывание	DF	Желтый
11	*Минотавр*	Остановка	.....	.....

Мы видим, что в данном случае Минотавр достижим. Выделив в предпоследнем столбце те из коридоров, которые остались желтыми (в соответствии с показаниями последнего столбца), получим следующий простой путь, ведущий от A к F:

$$AB,\ BC,\ CD,\ DF.$$

Пример 2. Если же поиск начинается с площадки K, то процесс поиска можно изобразить в схеме, представленной на таблице 2.

Мы видим, что в этом случае Минотавр не достижим.

Перейдем теперь к доказательству утверждений 1—3.

Доказательство утверждения 1. Покажем сперва методом индукции по числу ходов Тезея, что на любом этапе процесса поиска возникает следующая альтернатива (т. е. имеет место один из следующих двух случаев, взаимно исключающих друг друга):

Таблица 2

Порядковый номер хода	Каким признаком руководствуется Тезей	Ход	Пройденный коридор	Цвет коридора после прохождения
1	*Зеленая улица*	Разматывание	*KN*	Желтый
2	» »	»	*NL*	»
3	» »	»	*LM*	»
4	» »	»	*MN*	»
5	*Петля*	Наматывание	*NM*	Красный
6	*Пятый случай*	»	*ML*	»
7	» »	»	*LN*	»
8	» »	»	*NK*	»
9	*Ариадна*	Остановка	.....	.....

а) В лабиринте нет желтых коридоров; при этом Тезей находится на площадке A (Ариадны).

б) В лабиринте имеются желтые коридоры, причем они, будучи рассмотрены в том порядке, в каком они были пройдены Тезеем, образуют путь, ведущий от A до площадки, на которой находится Тезей.

Вместе с тем будет обнаружено, что Тезей никогда не проходит по красному коридору.

Высказанные положения являются очевидными для начала процесса, когда Тезей еще находится на площадке A и никакой коридор еще не пройден (все коридоры — зеленые). Предположим теперь, что указанные альтернативы справедливы после $(n-1)$-го хода и покажем, что тогда они справедливы и после n-го хода (разумеется, если только $(n-1)$-й ход еще не привел к остановке).

Пусть после $(n-1)$-го хода имеет место случай *а*. Тогда очередной ход может быть либо продвижением по зеленому коридору от A до некоторой смежной площадки K (после n-го хода возникает случай *б* с единственным желтым коридором AK), либо остановкой на площадке A (после n-го хода сохраняется случай *а*).

Допустим теперь, что после $(n-1)$-го хода имеет место случай *б* с s желтыми коридорами, образующими путь AA_1, A_1A_2, ..., $A_{s-1}K$. В зависимости от того, каким признаком руководствуется Тезей при выборе очередного n-го хода, после n-го хода имеются следующие возможности:

1. *Минотавр.* Происходит остановка на площадке K с сохранением прежних желтых коридоров (случай *б* после n-го хода).

2. *Петля.* Тезей наматывает нить, т. е. отступает по желтому коридору KA_{s-1}, который теперь уже становится красным. Желтый путь укорачивается на один коридор; если число s коридоров в прежнем пути было больше 1, то после n-го хода будет иметь место случай *б* с $(s-1)$ желтыми коридорами; если же $s=1$, то будет иметь место случай *а*.

3. *Зеленая улица.* Тезей разматывает нить, т. е. продвигается по зеленому коридору, который теперь уже становится желтым. Имеет место случай *б* с $s+1$ желтым коридором.

4. *Ариадна.* Этим признаком Тезей не будет руководствоваться, ибо если он даже и вернется на площадку Ариадны, идя по желтому пути

$$AA_1,\ A_1A_2,\ \ldots,\ A_{s-1}K$$

(т. е. если $K=A$), то в соответствии с принятым методом поиска он должен руководствоваться признаком *петля*.

5. При отсутствии первых четырех признаков происходит наматывание нити, которое приводит так же, как и в случае петли, к случаю *а* (при $s=1$) или к случаю *б* (при $s>1$).

Таким образом, полностью установлена упомянутая выше альтернатива, а также выяснено, что по каждому коридору Тезей проходит не более двух раз (по красному он никогда не проходит). Поскольку, кроме того, число всех коридоров в лабиринте конечно, то и последовательность ходов должна быть конечной; но эта последовательность может оборваться только при остановке Тезея на площадке Минотавра или на площадке Ариадны.

Доказательство утверждения 2. В случае остановки на площадке Минотавра факт достижимости очевиден, причем нить Ариадны протянута вдоль желтого пути, существование которого было установлено выше. Отсутствие петель в нити обеспечивается тем, что каждый раз, когда в процессе блуждания Тезей обнаруживает петлю, он возвращается обратно и тем самым ликвидирует ее.

Доказательство утверждения 3. Заметим, что в случае остановки на площадке Ариадны

1) каждый коридор лабиринта либо пройден дважды (красный коридор), либо ни разу не пройден (зеленый коридор); иными словами, нить полностью намотана (желтых коридоров в лабиринте нет) *).

*) Доказательство этих фактов предоставляется читателю.

2) все коридоры, выходящие из A, являются красными, ибо в соответствии с принятым соглашением признак *Ариадна* учитывается лишь в том случае, когда предшествующие ему в схеме три признака, среди которых и *зеленая улица*, не имеют места.

Предположим теперь от противного, что Минотавр достижим и пусть AA_1, A_1A_2, ..., A_nM — путь, ведущий от A к M. Первый коридор в этом пути является красным, ибо он выходит из A, последний же является зеленым, ибо Тезей в своих поисках до Минотавра не добрался. Пусть A_iA_{i+1} — первый зеленый коридор в этой последовательности. Таким образом, к A_i примыкают зеленые и красные коридоры. Рассмотрим теперь последнее прохождение Тезея через площадку A_i; оно, очевидно, произошло по одному из красных коридоров, примыкающих к A_i, и могло быть только наматыванием нити, ибо оно было уже повторным прохождением по коридору. Следовательно, оно было вызвано либо наличием петли, либо пятым случаем, т. е. отсутствием всех четырех признаков. Однако последнего быть не может, ибо из A_i выходит зеленый коридор A_iA_{i+1}. Поэтому приходится допустить, что последнее прохождение через A_i было связано с обнаружением петли. Однако это немедленно приводит к противоречию, чем и завершается доказательство утверждения 3. Именно, если бы в A_i была обнаружена петля, то это означало бы, что из нее выходят по крайней мере два желтых коридора; сделав свой очередной ход, Тезей превратил бы один из этих желтых коридоров в красный, оставшийся же желтый коридор должен быть обязательно пройден впоследствии повторно, ибо желтых коридоров не должно оставаться; тогда же будет пройдена еще раз площадка A_i. Это противоречит допущению, что рассматривается последнее прохождение через A_i.

Сделаем теперь следующее замечание. В предложенном нами предписании для поиска пути допускается известный элемент случайности. Именно, при условии *зеленая улица* очередной ход не определен однозначно, поскольку с данной площадки могут оказаться выходы в несколько зеленых коридоров, а наше предписание не предусматривает, какой из них нужно выбрать, точнее — оно допускает произвольный случайный выбор одного из них. Тем самым нарушается свойство детерминированности, о котором в предыдущем параграфе говорилось, что оно присуще всем алгоритмам. Этот элемент случайности можно было бы, однако,

легко устранить (и тем самым превратить рассмотренное предписание в алгоритм) хотя бы соглашением, что при наличии нескольких зеленых коридоров выбирается коридор по некоторому правилу; например, выйдя на площадку, Тезей начинает ее обходить по часовой стрелке до появления первого зеленого коридора и далее следует по нему, или же каким-нибудь другим соглашением. Рассмотрение таких предписаний, в которых заранее предусматриваются акты случайного выбора, представляет большой теоретический и практический интерес, особенно в современной теории игр. Однако мы не будем касаться этого вопроса, а будем рассматривать лишь строго детерминированные процессы.

В заключение полезно сравнить обе задачи данного параграфа и выявить определенную взаимосвязь между ними. При всем их внешнем различии эти задачи по своему существу весьма родственны, точнее — первая является частным случаем второй. Действительно, если каждой позиции в игре «15» сопоставить площадку, а каждому переходу от одной позиции в смежную позицию — коридор, соединяющий соответствующие площадки, то первая задача данного параграфа становится задачей поиска пути в лабиринте частного вида с 16! площадками, каждая из которых сообщается через коридоры с двумя, тремя, или четырьмя смежными площадками. Но тогда предложенный нами алгоритм поиска пути может быть перенесен и на игру «15». Легко понять, что он представляет собою попросту некоторый усовершенствованный вариант алгоритма переборки; усовершенствование заключается в том, что в переборке позиций и передвижении не допускаются излишние повторения более, чем два раза.

Для лабиринта частного случая (соответствующего игре «15») удалось найти и более простой алгоритм.

Вместе с тем представляется вполне естественным полагать, что в общем случае, когда алгоритм должен быть пригодным для любого произвольного лабиринта, он не может быть ничем иным как некоторым видом переборки. Поэтому вряд ли можно надеяться на создание алгоритма более простого, чем тот, который был предложен нами.

§ 3. ПРОБЛЕМА СЛОВ

Проблема слов представляет собою далеко идущее обобщение игры «15» и поисков Тезея. Если игра «15» сводится к поискам в специальном, заранее фиксированном конечном лабиринте, а поиски Тезея могут происходить

в произвольном конечном лабиринте, то проблема слов является в определенном смысле проблемой поиска в бесконечном лабиринте. Проблема слов возникла в специальных разделах современной алгебры, называемых *теорией ассоциативных систем и теорией групп*, однако ее значение выходит за рамки этих специальных теорий. Разные варианты этой проблемы были с успехом исследованы выдающимися советскими математиками Андреем Андреевичем Марковым и Петром Сергеевичем Новиковым, а также их учениками.

Введем сперва некоторые предварительные понятия:

Алфавитом будем называть любую конечную систему попарно различных знаков, называемых буквами этого алфавита. Так, например, можно рассматривать алфавит $\{\alpha, ц, z, ?\}$, состоящий из греческой буквы α, русской $ц$, латинской z и вопросительного знака. *Словом* в данном алфавите называется любая последовательность букв этого алфавита. Например, *abaa* и *bbac* являются словами в алфавите $\{a, b, c, \}$. Если слово L является частью слова M, то будем говорить о *вхождении* слова L в слово M. Так, например, в слове *abc bc bab* имеются два вхождения слова *bcb*, одно начиная со второй буквы, а другое — с четвертой. Мы будем рассматривать преобразования одних слов в другие посредством некоторых допустимых подстановок, которые задаются в виде

$$P \text{---} Q \text{ или } P \longrightarrow Q,$$

где P, Q — два слова в том же алфавите.

Применение *ориентированной подстановки* $P \longrightarrow Q$ к слову R возможно в том случае, когда в нем имеется хотя бы одно вхождение левой части P; оно заключается в замене любого одного такого вхождения соответствующей правой частью Q. Применение же *неориентированной подстановки* $P \text{---} Q$ допускает как замену вхождения левой части правой, так и замену вхождения правой части левой. Начиная с этого места, мы будем рассматривать преимущественно неориентированные подстановки, называя их просто подстановками, там, где это не вызывает недоразумений.

Пример 1. Подстановка $ab \text{---} bcb$ применима четырьмя способами к слову *abcbcbab*; замена каждого из двух вхождений *bcb* дает слова:

$$a\,ab\,cb\,ab, \quad abc\,\underline{ab}\,ab,$$

а замена каждого из двух вхождений ab дает слова:

$$\underline{bcb}\,cbc\,bab,\quad abc\,bcb\,\underline{bcb}.$$

К слову же $bacb$ эта подстановка не применима.

Условимся называть *ассоциативным исчислением* совокупность всех слов в некотором алфавите вместе с какой-нибудь конечной системой допустимых подстановок.

Для того чтобы задать ассоциативное исчисление, достаточно указать соответствующие алфавит и систему подстановок.

Если слово R может быть преобразовано в слово S посредством однократного применения допустимой подстановки, то и S может быть преобразовано в R таким же путем; в таком случае R и S будем называть *смежными* словами. Последовательность слов

$$R_1,\ R_2,\ldots,\ R_{n-1},\ R_n$$

таких, что R_1 и R_2 смежны, R_2 и R_3 смежны, ... R_{n-1} и R_n смежны, будем называть *дедуктивной цепочкой*, ведущей от R_1 к R_n. Если существует дедуктивная цепочка, ведущая от слова R к слову S, то, очевидно, существует и дедуктивная цепочка, ведущая от S к R; в таком случае слова будем называть *эквивалентными* и обозначать это так: $R \sim S$. Ясно также, что если $S \sim R$ и $R \sim T$, то $S \sim T$. В дальнейшем нам понадобится еще следующая теорема:

Теорема. *Пусть* $P \sim Q$; *тогда, если в каком-либо слове* R *имеется вхождение* P, *то в результате подстановки вместо него* Q *получается слово, эквивалентное* R.

Доказательство. В условиях теоремы слово R удобно представить в виде SPT, где S — часть слова R, предшествующая вхождению P, и T — часть, следующая за ним, а преобразованное слово в виде SQT. Далее, в силу эквивалентности $P \sim Q$ существует дедуктивная цепочка

$$P,\ P_1\ P_2,\cdots,\ P_m,\ Q.$$

Но в таком случае, как легко видеть, последовательность слов

$$SPT,\ SP_1T,\ SP_2T,\cdots,SP_mT,\ SQT$$

является дедуктивной цепочкой, ведущей от SPT (т. е. R) к SQT (т. е. к преобразованному слову). Теорема доказана.

Пример 2. Рассмотрим ассоциативное исчисление, которое было изучено Г. С. Цейтиным.

Алфавит:

$$\{a,\ b,\ c,\ d,\ e\}.$$

Система допустимых подстановок:

$$\begin{gathered} ac \longrightarrow ca \\ ad \longrightarrow da \\ bc \longrightarrow cb \\ bd \longrightarrow db \\ abac \longrightarrow abacc \\ eca \longrightarrow ae \\ edb \longrightarrow be \end{gathered}$$

В этом исчислении к слову *abcde* применима только третья подстановка и оно имеет только одно смежное слово *acbde*. Далее имеет место эквивалентность: $abcde \sim cadedb$, как видно из следующей дедуктивной цепочки:

$$\underline{ab}cde,\ \underline{ac}bde,\ \underline{ca}bde,\ \underline{cad}be,\ cadedb.$$

Если же взять слово *aaabb*, то к нему не применима ни одна из подстановок и поэтому оно не имеет никаких смежных слов; тем более не существует слов, отличных от *aaabb*, которые были бы ему эквивалентны.

Для каждого ассоциативного исчисления возникает своя специальная *проблема эквивалентности слов*. Она заключается в следующем:

Для любых двух слов в данном исчислении требуется узнать, эквивалентны они или нет.

Поскольку различных слов в любом исчислении найдется бесчисленное множество, то фактически здесь имеется бесконечная серия однотипных задач, а решение мыслится в виде алгоритма, распознающего эквивалентность или неэквивалентность любой пары слов.

Может создаться впечатление, что проблема слов представляет собой искусственно надуманную головоломку, и, следовательно, нахождение такого алгоритма не представляет собою особого практического или теоретического интереса. На самом же деле это далеко не так; можно показать, что эта проблема является весьма естественной и имеет

большое теоретическое и практическое значение, вполне оправдывающее усилия, направленные на построение соответствующего алгоритма. Однако на данной стадии изложения мы воздержимся пока от обсуждения этого вопроса по существу и перейдем к рассмотрению некоторых конкретных фактов.

Во-первых, укажем на связь проблемы эквивалентности слов с проблемой Тезея. Если построить для каждого слова свою «площадку» и для каждой пары смежных слов — коридор, соединяющий соответствующие площадки, то ассоциативное исчисление предстанет перед нами в виде лабиринта с бесконечным числом площадок и коридоров, в котором от каждой площадки расходится только конечное число коридоров (правда, здесь возможны и такие площадки, из которых не выходит ни одного коридора: см. слово *aaabb* в примере 2). При этом дедуктивная цепочка, ведущая от какого-либо слова R к слову Q, будет представлять собою путь в лабиринте, ведущий от одной площадки к другой, а значит эквивалентности слов соответствует взаимная достижимость одной площадки от другой. Наконец, при такой трактовке сама проблема слов становится проблемой поиска пути в бесконечном лабиринте.

Для того чтобы лучше уяснить специфику тех трудностей, которые здесь возникают, рассмотрим предварительно *ограниченную проблему слов*, которая заключается в следующем:

Для любых двух слов R и T в данном ассоциативном исчислении требуется узнать, можно ли преобразовать одно в другое последовательным применением допустимых подстановок не более чем k раз (k — произвольное, но фиксированное натуральное число).

При такой постановке проблемы алгоритм строится легко: именно, можно пользоваться уже известным нам алгоритмом переборки, при котором просматривается список всех слов, начиная со слова R, переходя ко всем его смежным словам, потом к смежным для смежных и так далее k раз. Ответ на поставленный вопрос будет положительным или отрицательным в зависимости от того, будет ли обнаружено слово T в этом списке, или нет.

Однако если вернуться к неограниченной проблеме слов, то там положение существенно иное. Поскольку длина дедуктивной цепочки, ведущей от R к T (если такая существует), может оказаться сколь угодно большой, то, вообще говоря, неизвестно, когда следует считать законченным

процесс переборки. Пусть, например, мы уже продолжили процесс переборки до $10^{20} = 100\,000\,000\,000\,000\,000\,000$ и уже располагаем списком всех слов, которые можно получить из R при помощи многократных применений подстановок, общее число которых не превосходит 10^{20}, и пусть в этом списке слова T не оказалось. Дает ли это нам какое-либо основание для заключения о неэквивалентности слов R и T? Разумеется, нет, ибо не исключена возможность, что R и T эквивалентны, но минимальная дедуктивная цепочка, соединяющая их, еще длиннее.

Для получения желаемых результатов здесь приходится отказаться от простой переборки, здесь необходимо привлекать другие идеи, основанные на анализе самого механизма преобразования одних слов в другие посредством допустимых подстановок. Попытаемся, например, выяснить, эквивалентны ли в исчислении Цейтина (см. пример 2) слова *abaacd* и *acbdad*? Отрицательный ответ на этот вопрос вытекает из следующих соображений: в каждой из допустимых подстановок этого исчисления левая и правая части содержат одно и тоже число вхождений буквы a (или же вовсе не содержат этой буквы); поэтому в любой дедуктивной цепочке все слова должны содержать одно и то же число вхождений буквы a. Поскольку в предложенных двух словах число вхождений буквы a не одинаково, то эти слова не эквивалентны.

Обнаружение подобных *дедуктивных инвариантов*, т. е. свойств, остающихся неизменными для всех слов дедуктивной цепочки, и позволяет в некоторых случаях находить искомые разрешающие алгоритмы.

Пример 3.

Алфавит

$$\{a, b, c, d, e\}.$$

Система допустимых подстановок:

$$ab \to ba;\quad ae \to ea;\quad be \to eb;\quad de \to ed;$$
$$ac \to ca;\quad bc \to cb;\quad cd \to dc;$$
$$ad \to da;\quad bd \to db;\quad ce \to ec.$$

Эти допустимые подстановки не изменяют количество вхождений каждой буквы в слове, меняется только порядок букв в слове. Непосредственно ясно, что два слова эквивалентны в том и только в том случае, когда в них одинаковое число

вхождений каждой буквы. Поэтому алгоритм распознавания эквивалентности весьма прост и сводится к подсчету числа вхождений букв в каждой из этих слов и сравнению этих чисел.

Мы разберем ниже подробно более сложный пример, но предварительно условимся в следующем обобщении понятий «слово» и «допустимая подстановка». Именно, будем рассматривать, кроме обычных слов, в данном алфавите и *пустое слово*, не содержащее ни одной буквы, и примем для него обозначение Λ. Вместе с тем будем допускать и подстановки вида

$$P \text{—} \Lambda.$$

При этом замена левой части пустым словом означает просто, что из преобразуемого слова выбрасывается вхождение слова P. Замена же правой части левой означает, что между двумя какими-либо буквами преобразуемого слова или впереди его, или за ним вставляется слово P.

Пример 4. Дано ассоциативное исчисление в алфавите $\{a, b, c\}$ с системой подстановок:

$$b \text{—} acc$$
$$ca \text{—} accc$$
$$aa \text{—} \Lambda$$
$$bb \text{—} \Lambda$$
$$cccc \text{—} \Lambda$$

Требуется найти разрешающий алгоритм для проблемы эквивалентности слов в этом исчислении.

Построим один вспомогательный алгоритм — алгоритм приведения, который указывает для любого слова эквивалентное ему слово специального вида — приведенное слово. Для этого рассмотрим упорядоченную систему ориентированных подстановок

$$b \longrightarrow acc$$
$$ca \longrightarrow accc$$
$$aa \longrightarrow \Lambda$$
$$cccc \longrightarrow \Lambda,$$

и условимся, что в применении к какому-либо слову R алгоритм работает так: берется первая по порядку подстановка (ориентированная) из этого списка, которая применима

к слову R и при наличии нескольких вхождений ее левой части в слове R подстановку применяют к первому, самому левому, вхождению; для полученного таким образом слова R' опять берется первая в этом списке подстановка, которая к нему применима, и она тоже применяется к самому левому вхождению своей левой части; если после конечного числа таких шагов получается слово S, к которому ни одна из этих подстановок уже не применима, то говорят, что алгоритм применим к слову R и перерабатывает его в слово S*).

Мы покажем, что каково бы ни было слово R, алгоритм приведения применим к нему и перерабатывает его в одно из следующих восьми слов (приведенных слов):

$$\Lambda,\ c,\ cc,\ ccc,\ a,\ ac,\ acc,\ accc.$$

Действительно, если в слове R имеются вхождения буквы b, то сначала будет применяться первая подстановка, исключающая b и заменяющая его на acc, до тех пор, пока не останется ни одного b. Далее заработает вторая подстановка, перемещающая букву a левее буквы c до тех пор, пока не останется ни одного a, непосредственно следующего правее буквы c, иными словами, пока все a не окажутся впереди всех c. Наконец, начнется процесс вычеркивания пар соседних a и четверок соседних c, до тех пор пока не появится слово, содержащее не более одного a и не более трех c; таких же слов может быть всего восемь и они как раз и были перечислены выше.

Очевидно, каждое слово эквивалентно своему приведенному слову, поэтому два слова эквивалентны в том и только в том случае, когда им соответствует одно и то же приведенное слово, или два эквивалентных приведенных слова. Мы докажем ниже, что все 8 приведенных слов попарно неэквивалентны. Отсюда будет следовать, что два слова эквивалентны в том и только в том случае, когда им соот-

---

*) Алгоритмы для переработки слов в некотором алфавите, которые задаются таким образом посредством какой-либо упорядоченной системы ориентированных подстановок, называются нормальными. Теория нормальных алгоритмов, представляющая большой теоретический и практический интерес, создана членом-корреспондентом Академии наук СССР А. А. Марковым и изложена в его книге „Теория алгоритмов".

ветствует одно и то же приведенное слово. Вместе с тем будет построен и алгоритм для поставленной проблемы слов. Он будет заключаться в применении алгоритма приведения к каждому из двух рассматриваемых слов с последующим сравнением получаемых приведенных слов.

Пусть, например, даны слова *cacb* и *bb*. Находим приведенные слова:

1) *cacb*, *cacacc*, *acccacc*, *acccacccc*, *accacccccc*, *acacccccccc*, *aacccccccccccc*, *ccccccccccccc*, *cccccccccc*, *ccccc*, *cc*;

2) *bb*, *accb*, *accacc*, *acacccc*, *aacccccc*, *ccccccc*, *cccc*, Λ.

Вывод: слова *cacb* и *bb* неэквивалентны, ибо получились два различных приведенных слова: *cc* и Λ.

Доказательство попарной неэквивалентности восьми приведенных слов. Во-первых, заметим, что если дана дедуктивная цепочка, ведущая от какого-либо слова R, не содержащего буквы b, к слову S, также не содержащему этой буквы, то можно из нее извлечь дедуктивную цепочку, ведущую от R к S, но такую, что в промежуточных словах цепочки не встречается буква b. Действительно, если во всех словах данной дедуктивной цепочки заменить каждое вхождение буквы b словом *acc*, то получится последовательность слов, в которой каждые два рядом стоящие слова являются смежными (в смысле исчисления) или просто одинаковыми. Если удалить теперь лишние повторяющиеся слова (из рядом стоящих), то получится нужная дедуктивная цепочка. В дедуктивных цепочках этого типа совсем не участвует подстановка b — *acc*.

Далее, в каждой из оставшихся допустимых подстановок число вхождений буквы a в левой части и число вхождений в правой части либо одновременно четны, либо одновременно нечетны. Аналогичное утверждение справедливо и для буквы c. Значит — четность числа вхождений буквы a (или буквы c) является дедуктивным инвариантом для дедуктивных цепочек вышеуказанного вида. Отсюда сразу вытекает, что ни одно из четырех приведенных слов, содержащих по одному вхождению буквы a, не эквивалентно ни одному из четырех приведенных слов, не содержащих вовсе таких вхождений. Точно так же ни одно из приведенных слов, содержащих одно или три вхождения буквы c, не эквивалентно какому-либо слову, содержащему два таких вхождения или не содержащему ни одного вхождения.

Поэтому остается еще убедиться в неэквивалентности следующих пар слов:

$$\Lambda,\ cc;\ c,ccc;\ a,acc;\ ac,accc.$$

Если бы имела место эквивалентность хотя бы одной из первых трех пар, то, как следствие из теоремы этого параграфа, имела бы место и эквивалентность четвертой. Достаточно поэтому установить неэквивалентность пары ac, $accc$; и к этому мы сейчас и переходим.

Условимся в следующих терминах.

Индексом какого-либо *вхождения буквы* a в слове R называется число всех вхождений буквы c, встречающихся правее этого вхождения буквы a. *Индексом слова* R называется сумма индексов всех вхождений буквы a*). Каждая из подстановок $aa \to \Lambda$ и $cccc \to \Lambda$ не меняет четности индекса слова. Действительно, при подстановке aa вместо пустого слова индекс слова увеличивается на сумму равных между собою индексов этих двух вхождений буквы a, т. е. на четное число; при замене же вхождения aa пустым словом индекс слова уменьшается на четное число.

При подстановке $cccc$ индексы некоторых вхождений a увеличиваются на четыре, индексы других же не изменяются; в целом индексе слово увеличивается на четное число. Аналогично при зачеркивании $cccc$. Подстановка $b \to acc$ очевидным образом не меняет четности индекса слова.

Наконец, покажем, что подстановка $ca \to accc$ изменяет четность индекса слова. Для этого сравним слова

$$PcaQ \quad \text{и} \quad PacccQ;$$

индекс каждого вхождения a в часть P слова R изменяется ровно на 2, а индекс вхождения a в Q не изменяется. Что же касается индекса единственного вхождения a между P и Q, то он изменяется в точности на 3. В целом индекс слова изменяется на нечетное число.

Слова $'ac$ и $accc$ оба имеют индексы одной и той же четности; поэтому, если они эквивалентны, то в соответствующей дедуктивной цепочке (можно считать, что в ней буква b не встречается, см. выше) может быть лишь четное число подстановок $ca \to accc$.

---

*) Например, в слове $acbca$ первое слева вхождение буквы a имеет индекс 0 (правее нет вхождений буквы c); второе же слева вхождение буквы a имеет индекс 2. Индекс слова — 2.

Однако такое предположение приводит к противоречию, как видно из следующих рассуждений. Каждое применение подстановки $ca \to ccc$ меняет число вхождений буквы c на 2, поэтому четное число ее применений меняет число вхождений буквы c на число, кратное четырем. Очевидно, подстановка $cccc \to \Lambda$ меняет число вхождений буквы c в точности на 4, а подстановка $aa \to \Lambda$ совсем не меняет этого числа. Подведя итог всему сказанному, мы должны заключить, что если $ac \sim ccc$, то количества вхождений буквы c в них отличаются на число, кратное четырем, чего на самом деле нет. Итак, слова ac и ccc не эквивалентны. Вместе с тем мы завершили обоснование предложенного алгоритма.

Приведенное в примере 4 подробное решение проблемы слов для конкретного ассоциативного исчисления во многом характеризует понятия и методы, возникающие и при исследовании проблемы слов для других исчислений. Нам остается еще пояснить содержательный смысл проблемы слов и ее значение в современной алгебре. Это удобно сделать на конкретном примере.

Возьмем какой-нибудь квадрат (рис. 3, I). Рассмотрим следующие три его *самосовмещения*, т. е. преобразования, которые переводят квадрат в самого себя:

1) симметрия (зеркальное отображение) относительно вертикальной оси, проходящей через центр квадрата O;

2) симметрия относительно горизонтальной оси, проходящей через центр квадрата O;

3) вращение по часовой стрелке на 90° вокруг центра O.

Эти преобразования будем называть элементарными и обозначать буквами a, b, c соответственно. На рис. 3 (III—IV—V) показано, как изменяется расположение вершин квадрата (II) при каждом из элементарных преобразований.

Заметим, что в результате последовательного осуществления каких бы то ни было двух или нескольких самосовмещений квадрата происходит также самосовмещение квадрата. Будем придерживаться общепринятого определения, согласно которому умножить два заданных преобразования (в частности — два самосовмещения квадрата) — значит последовательно произвести их одно за другим. Условимся еще для операции умножения преобразований сохранить обычную систему обозначений, а также термин *произведение* для результирующего преобразования. Так, например, произведение cc есть результат двух последовательных вращений

по 90°, т. е. вращение на 180°; произведение ac — результат симметрии относительно вертикальной оси с последующим поворотом на 90° — равносильно симметрии относительно левой диагонали (см. рис. 3, I). Произведение же $(ac)(cc)$ двух предыдущих произведений равносильно симметрии относительно правой диагонали (см. рис. 3, I).

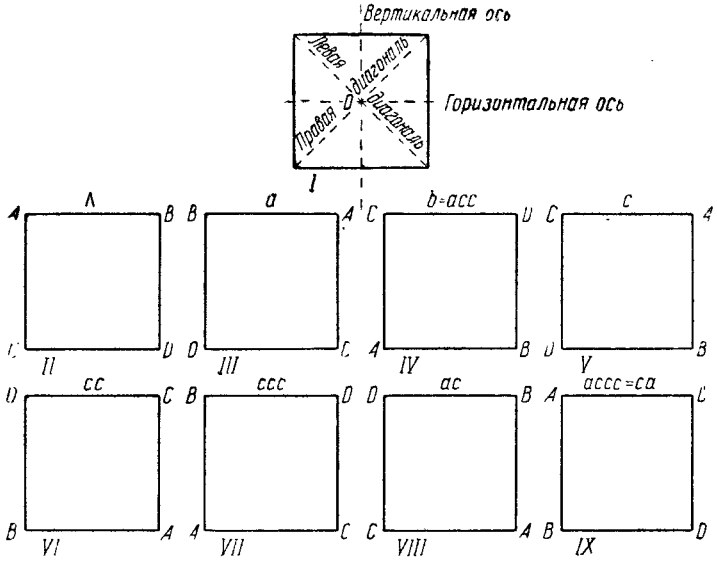


Рис. 3.

Наше умножение не является переместительным; на рисунке 3 (VIII—IX) изображены два различных расположения вершин квадрата, которые возникают при самосовмещениях: ac, ca исходного квадрата (II). Однако его роднит с арифметическим умножением свойство ассоциативности (сочетательности): каковы бы ни были преобразования p, q, r, имеет место тождество $(pq)r = p(qr)$. Благодаря этому в произведениях можно пренебрегать расстановкой скобок; так, например $(ac)(cc)$ и $(((ac)c)c)$ дают одно и то же самосовмещение квадрата, а именно симметрию относительно левой диагонали.

Объектом наших последовательных рассмотрений будет совокупность Ω, состоящая из элементарных преобразований a, b, c и всех самосовмещений квадрата, которые могут

быть представлены как произведения конечного (но произвольного) числа элементарных преобразований. Благодаря ассоциативности умножения при символической записи элементов из Ω можно опустить скобки и ограничиться записью в соответствующем порядке букв, изображающих соответствующие элементарные самосовмещения, например *abb*, *cabb*, *accc* и т. д. Это означает, что всякое произведение изобразится в виде слова в алфавите $\{a, b, c\}$.

Из ассоциативности умножения вытекает также, что если к слову P приписать справа слово Q так, чтобы получилось одно слово PQ, то оно будет изображать произведение самосовмещений, изображенных словами P и Q соответственно; так, например, слово *abccab* изображает произведение самосовмещений, изображаемых словами *abc* и *cab*.

Очевидно, графически различных (т. е. различных по своему начертанию) слов в алфавите $\{a, b, c\}$ существует бесчисленное множество; однако графически различные слова могут изображать одно и то же самосовмещение из Ω. В этом случае естественно слова считать равными и обозначать такое равенство обычным образом. Читатель легко проверит справедливость равенств:

$$b = acc, \tag{1}$$

$$ca = accc. \tag{2}$$

Для этого достаточно сравнить расположения вершин квадрата, которые возникают в результате преобразований из левой и правой частей этих равенств. Далее, легко видеть, что каждое из слов: *aa*, *bb*, *cccc* задает одно и то же самосовмещение, а именно так называемое тождественное преобразование, при котором все вершины остаются на прежних местах. Поскольку это преобразование ничего не меняет, целесообразно его изображать также пустым словом $\wedge$. Итак, имеют место и равенства

$$aa = \wedge, \tag{3}$$

$$bb = \wedge, \tag{4}$$

$$cccc = \wedge. \tag{5}$$

Сопоставление равенств (1)—(5) с допустимыми подстановками ассоциативного исчисления примера 4 подсказывает следующее предложение, устанавливающее связь между этим исчислением и рассматриваемой системой преобразований квадрата:

Два произведения элементарных самосовмещений квадрата задают одно и то же преобразование в том и только в том случае, когда изображающие их слова эквивалентны в исчислении примера 4.

В самом деле из равенств (1)—(5) вытекает, что при каждом применении любой допустимой подстановки к любому слову S оно переходит в равное слово. Например, применяя подстановку $ca \to accc$ к слову $bcac$, получаем слово $bacccc$; но ведь в силу ассоциативности умножения мы можем писать: $bcac = b(ca)c$ и $bacccc = b(accc)c$; правые части равны как произведения соответственно равных множителей, значит, и левые равны между собой. Итак, *любые два смежных слова равны.*

Теперь уже легко понять, что эквивалентность двух слов в нашем ассоциативном исчислении влечет за собой их равенство (т. е. одинаковость самосовмещений, задаваемых ими). Действительно, если $S \sim T$, то в соответствующей цепочке всякие два смежных элемента равны, а значит, и $S = T$.

Имеет место и обратное утверждение: *если слова равны, то они эквивалентны.* Действительно, если два слова равны, то равны и соответствующие им приведенные слова (это вытекает из прямого утверждения). Вместе с тем непосредственно можно проверить, что все восемь приведенных слов задают попарно различные самосовмещения (см. рисунки 3, II—IX, где изображены расположения вершин квадрата (рис. 3, II) при самосовмещениях, соответствующих восьми приведенным словам). Поэтому, если два слова равны, то им соответствует одно и то же приведенное слово, а, значит, по ранее доказанному они эквивалентны.

Таким образом, формальная эквивалентность двух слов в нашем исчислении получает конкретный геометрический смысл и распознавание эквивалентности двух слов приобретает значение решения конкретной геометрической задачи. Вместе с тем описанный нами алгоритм предстает перед нами как общий метод решения любой геометрической задачи данного типа.

Аналогично обстоит дело и с другими исчислениями, в которых формальная эквивалентность также допускает конкретное геометрическое, алгебраическое или иное истолкование. Без преувеличения можно сказать, что во всякой области математики имеются теоремы, которые могут быть сформулированы после некоторой обработки в виде утвержде-

ний об эквивалентности двух слов в некотором исчислении. В этой книжке нет возможности разобрать этот круг вопросов; некоторые пояснения будут даны еще по ходу дальнейшего изложения (см. § 6).

Заметим еще, что, исходя из того геометрического истолкования, которое мы дали проблеме слов в рассмотренном исчислении, можно теперь построить алгоритм непосредственно и даже несколько проще. Именно, достаточно для каждого из двух предлагаемых произведений фактически осуществить последовательность соответствующих самосовмещений (хотя бы на чертеже) и сравнить результаты.

Упражнение. Решить проблему слов для ассоциативного исчисления, заданного в алфавите $\{a,b\}$ с допустимыми подстановками:

$$aaa = bb,$$

$$bbbb = \wedge.$$

§ 4. ВЫЧИСЛИТЕЛЬНАЯ МАШИНА С АВТОМАТИЧЕСКИМ УПРАВЛЕНИЕМ

Создание алгоритма для задач некоторого данного типа (и в особенности «хорошего», удобного алгоритма) в тех случаях, когда это удается, связано вообще с тонкими и сложными рассуждениями, требующими высокой квалификации и большой изобретательности. Однако с того момента, когда такой алгоритм уже создан, процесс решения соответствующих задач становится таковым, что его может в точности выполнять человек, не имеющий ни малейшего понятия о сущности самой задачи. Требуется лишь, чтобы этот человек был способен выполнять те простейшие и немногочисленные элементарные операции, из которых складывается процесс, и, кроме того, чтобы он добросовестно и беспрекословно руководствовался предложенным предписанием (алгоритмом). Такой человек, действуя, как иногда говорят, чисто машинально, все же успешно решал бы любую задачу рассматриваемого типа. Выражение «машинальное действие» употребляется здесь лишь для пояснения детерминированности алгоритма; однако при современном развитии науки и техники оно уже приобретает прямой смысл. Именно, на месте гипотетического человека, решающего задачу, смысла которой он не понимает (или не желает знать), можно действительно поставить машину, выполняющую

тот же процесс. Такой и является современная вычислительная машина с автоматическим управлением.

Наша ближайшая задача состоит в том, чтобы выяснить основные принципы устройства и работы подобных машин. Для этого мы предварительно еще раз обратимся к рассмотрению алгоритмического процесса, осуществляемого человеком (вычислителем).

Руководствуясь алгоритмом, человек-вычислитель совершает процесс, в котором происходит прием, хранение, переработка и выдача некоторых сведений (некоторой *информации*). Обычно эти сведения вычислитель записывает (изображает) на бумаге с помощью цифр, букв или других символов. Совокупность всех этих символов принято называть *алфавитом*. Так, например, в алгебре применяется алфавит, в который входят, кроме обычных букв, цифры, знаки алгебраических операций, скобки и т. д.

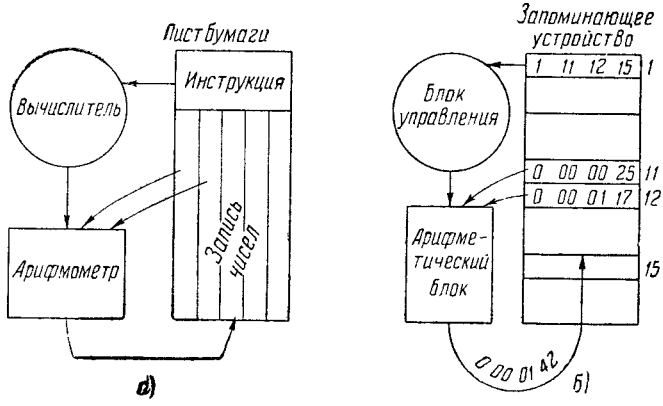


Рис. 4.

Для вычислительного процесса, происходящего с участием человека-вычислителя, характерно наличие следующих трех факторов (см. схему на рис. 4, *а*):

1. Хранение информации обеспечивается обычно записью всех сведений на листе бумаги, при этом к сведениям относят также инструкцию (схему алгоритма) для решения задачи. Заметим, что фактически вычислитель не записывает буквально все на бумаге; кое-что он просто запоминает (хранит не на листе бумаги, а в памяти), а некоторые сведения черпает из разных справочников и таблиц,

Однако эти обстоятельства не должны затушевывать основного факта, заключающегося в том, что вычислительный процесс предполагает наличие таких средств, которые обеспечивают хранение всех необходимых сведений. Таким образом, под листом бумаги в нашей схеме следует понимать совокупность всех средств, которые обеспечивают хранение сведений.

2. Обработка информации предполагает способность вычислителя выполнять отдельные элементарные операции, предусмотренные в алгоритме, и может осуществляться вычислителем посредством специализированных механизмов, например арифметические операции над числами могут быть реализованы на арифмометре. Каждая отдельная операция заключается в том, что вычислителем в соответствии с инструкцией извлекаются некоторые сведения (например, числа) из определенных граф листа бумаги, они подаются на обрабатывающее устройство (арифмометр), а результат обработки помещается во вполне определенное место листа.

3. Управление процессом, т. е. принятие решения о реализации на данном этапе процесса той или иной операции и создание условий для ее выполнения, осуществляется самим вычислителем согласно инструкции.

Из каких устройств составлена машина и как они взаимодействуют между собой? Ответ на этот вопрос подсказывается тем, что в ней должны происходить такие же процессы, какие были только что описаны, но уже без участия вычислителя.

Во-первых, для изображения информации машина также должна иметь определенный *алфавит*; однако вместо обычного графического изображения символов, различаемых друг от друга по их начертанию, в машине различные символы алфавита изображаются некоторыми различными друг от друга физическими состояниями, например, различными электрическими напряжениями, или различными состояниями намагничивания.

Целый ряд соображений оправдывает преимущественное применение алфавита из двух символов (*двоичного* алфавита), условно называемых 0 и 1. Так, например, этот алфавит проще всего осуществляется физически в электрических цепях в виде двух состояний: высокое напряжение (или — ток идет) и низкое напряжение (или — ток не идет). Далее приходится считаться и с тем, что простейшие логические операции совершаются над переменными, могущими

принимать два значения: «истинно» и «должно». Однако выбор того или другого алфавита и способ изображения в нем нужных сведений далеко не является решающим для понимания устройства и работы машины. Поэтому, ограничившись тем замечанием, что в современных машинах пользуются преимущественно двоичным алфавитом и двоичной системой счисления (вместо употребительной десятичной), мы в дальнейшем не будем вдаваться в эти детали.

Итак, информация, которая поступает в машину, а также та, которая вырабатывается в ней в процессе ее работы, изображена в виде некоторых физических параметров. В интересующих нас случаях все сведения, составляющие информацию, будут зашифрованы числами. В частности, в виде набора чисел будет зашифрован и сам алгоритм, которым должна «руководствоваться» машина в своей работе. Алгоритмы, составленные специально для машин, называют обычно *программами*. Программа является важнейшей частью информации, с которой имеет дело машина.

Далее, в соответствии со схемой рис. 4, *а* в машине имеются устройства (органы), выполняющие функции хранения, информации, ее обработки, управления процессом (см. схему на рис. 4, *б*).

1. Запоминающее устройство играет роль листа бумаги. На условном языке машины в нем фиксируются все нужные сведения, включая программу. Возможность физического осуществления органа, выполняющего такие функции, вряд ли вызывает у кого-либо сомнение. Действительно, эту функцию может выполнять магнитная лента, на которой кодированные сведения фиксируются и с которой они снимаются примерно так же, как на обычном магнитофоне. Запоминающее устройство (*память машины*) состоит из набора ячеек, занумерованных натуральными числами: 1, 2, 3.... Эти числа называются *адресами* ячеек. Каждая ячейка хранит или может принять на хранение одно закодированное сообщение; в вычислительных машинах всякое такое сообщение представлено, как мы уже отметили, в виде некоторого числа.

Практически в электронных машинах, кроме магнитной ленты, применяются и другие средства «запоминания», а именно электронно-лучевые трубки, принцип работы которых несколько напоминает работу телевизионных трубок, а также магнитный барабан, причем функции запоминания разделены между этими органами целесообразным способом. Однако

мы не будем различать отдельных составных частей памяти и без ущерба для понимания дела будем исходить из представления об едином запоминающем устройстве, хотя бы в виде магнитной ленты.

2. Арифметический блок играет ту же роль, что и обычный арифмометр, хотя физические принципы, лежащие в основе его конструкции, сильно отличаются от тех, которые использованы в обычном арифмометре. Переработка, подаваемых в него данных в нужный результат (например, сложение чисел) происходит путем преобразования в электронном устройстве входных электрических сигналов, изображающих входные данные, в электрические сигналы, изображающие выходные данные. Входные сигналы поступают в арифметический блок из ячеек памяти, где они хранились, а выходной сигнал отправляется из него в ту ячейку, в которой он будет храниться. Схематически это изображено на рис. 4, *в*, где числа из ячеек 11 и 12 складываются, а результат отправляется в ячейку 15. Для того чтобы такая операция была осуществлена в машине в некотором такте, необходимо, чтобы к началу этого такта были произведены соединения ячеек 11 и 12 с арифметическим блоком и арифметического блока с ячейкой 15; также необходимо, чтобы арифмометр был включен на нужную операцию (в данном случае на сложение). Все это входит уже в «компетенцию» блока управления.

3. Блок управления предназначен для выполнения функций, которые в схеме рис. 4, *а* выполняет сам вычислитель. Именно, на каждом этапе работы машины управляющее устройство создает условия для реализации очередной операции процесса. При этом оно действует как автоматическая телефонная станция, соединяющая тех «абонентов» (узлы и ячейки машины), которые участвуют в данной отдельной операции. Выражаясь образно, блок управления заглядывает в программу и в соответствии с нею отправляет распоряжения о срабатывании тех узлов машины, которые должны обеспечить очередную операцию.

Для более точного описания возникающей здесь картины укажем, что каждая машина характеризуется вполне определенной системой команд (приказов), которые она может принимать к исполнению. При этом всякая программа, вкладываемая в машину, представляет собою определенную комбинацию команд и некоторых вспомогательных чисел (параметров), которые помещаются в ячейки «памяти». Напри-

мер, в машине БЭСМ Академии наук СССР принята так называемая трехадресная система команд, каждая из которых является последовательностью четырех чисел:

$$\alpha\beta\gamma\delta,$$

из которых первое указывает номер предписываемой операции, последующие два — адреса двух ячеек, над содержимым которых совершается операция, а последнее — адрес ячейки, в которую следует поместить результат (всего три адреса).

Практически каждая команда записывается в одной ячейке в виде одного числа, цифры которого разбиваются на четыре группы, имеющие соответствующие назначения. Например, на рис. 4, *б* в ячейке 1 запоминающего устройства записано число 1 11 12 15, которое представляет собою следующий зашифрованный приказ:

«*Сложить* (*операция* № 1) *числа из ячеек* 11, 12 *и результат отправить в ячейку* 15».

(Здесь принято разделение на группы справа налево по два разряда. Для определенности мы будем и дальше придерживаться такого метода записи команд.)

Обычно система команд насчитывает несколько десятков команд. Мы укажем здесь только самые употребительные.

1. Арифметические команды:

а) 1 β γ δ — сложить число из β с числом из γ и сумму отправить в δ;

б) 2 β γ δ — вычесть из числа, хранящегося в β, число из γ и разность отправить в δ;

в) 3 β γ δ — умножить число из β на число из γ и произведение отправить в δ;

г) 4 β γ δ — разделить число из β на число из γ и частное отправить в δ.

2. Команды передачи управления:

д) 5 00 00 δ — переходить к команде, хранящейся в δ (*безусловная* передача управления);

е) 5 01 γ δ — переходить к команде, хранящейся в δ при условии, что в ячейке γ хранится положительное число;

ж) 5 02 γ δ — переходить к команде, хранящейся в δ при условии, что в ячейке γ хранится отрицательное число.

3. Команда остановки:

0 00 00 00.

Кроме перечисленных команд имеются еще и команды так называемых *логических операций* и другие, на которых мы здесь останавливаться не будем. Перечисленных команд вполне достаточно для составления из них самых разнообразных программ; некоторые примеры будут рассмотрены в следующем параграфе.

Команды *е* — *ж* называются *условными*; они принимаются к исполнению лишь в том случае, если выполнено соответствующее условие, в противном случае они пропускаются управляющим устройством без выполнения.

Обычно команды выполняются машиной в том порядке, в каком они записаны в ячейках памяти, причем отклонение от этого порядка может произойти только в результате команды о передаче управления (безусловной или условной, если выполнено соответствующее условие).

Работа машины распадается на *такты*, в течение каждого из которых выполняется одна очередная команда. К началу каждого такта из одной ячейки памяти поступает в блок управления содержащееся в этой ячейке число-команда. Как только команда попадает туда, управление осуществляет автоматически соответствующие соединения и тем самым обеспечивает выполнение очередной операции процесса. После этого в блок управления поступает следующая команда, и машина срабатывает на следующую операцию и т. д. до результативной остановки машины.

Возможность технической реализации такого блока управления не представляет собою ничего неожиданного; ведь от этого устройства требуется не больше того, что умеет делать любая автоматическая телефонная станция, в которой номер набирается путем посылки соответствующего электрического сигнала.

Таковы основные три устройства вычислительной машины. Правда, в ней имеется еще ряд важных органов, в частности, для ввода данных в машину, для вывода из нее выработанных ею результатов. Однако эти устройства не имеют значения при рассмотрении принципов работы машины и выяснении ее логических и математических возможностей. Поэтому мы можем и не привлекать их в дальнейшем к рассмотрению, предполагая, что подача информации в машину и снятие результативной информации производится непосредственно на запоминающем устройстве.

§ 5. ПРОГРАММА
(машинный алгоритм)

В этом параграфе мы рассмотрим примеры программ, составленных для электронной вычислительной машины с трехадресной системой команд. Эти программы получаются из рассмотренных ранее алгоритмов путем их переработки с учетом того, что их исполнителем будет теперь уже не человек, а автоматическая машина с рассматриваемой системой команд. Разбор предлагаемых примеров призван уяснить реальный смысл употребляемого обычно выражения *«машина руководствуется в своей работе введенной в нее программой»*. Именно, будет выяснено, каким образом регулируется тот или другой порядок поступления команд в блок управления и как удается при помощи сравнительно небольшого числа команд обеспечить зачастую очень длинную цепь операций, необходимых для решения того или другого варианта решаемой задачи.

В дальнейшем мы полагаем, что арифметические операции сложения, вычитания, умножения и деления указываются в командах номерами 1, 2, 3, 4 соответственно (см. § 4).

Пример 1. Запрограммируем решение системы уравнений

$$ax + by = c,$$
$$dx + ey = f.$$

Примем для определенности, что коэффициенты a, b, c, d, e, f помещены подряд в ячейки памяти, начиная с номера 51:

Адрес	Содержание
51	a
52	b
53	c
54	d
55	e
56	f

Далее, для промежуточных и окончательных результатов вычислений отведем ячейки 31—50.

Как видно из формул

$$x = \frac{ce - fb}{ae - bd}, \qquad y = \frac{af - cd}{ae - bd},$$

для получения результата нужно совершить последовательно 6 умножений, три вычитания и два деления. В соответствии с этим предлагаемая нами программа состоит из 12 команд, записанных в ячейках 1—12:

Адрес	Содержание (команды)
1	3 53 55 31
2	3 56 52 32
3	3 51 56 33
4	3 53 54 34
5	3 51 55 35
6	3 52 54 36
7	2 31 32 37
8	2 33 34 38
9	2 35 36 39
10	4 37 39 40
11	4 38 39 41
12	0 00 00 00

Команды поступают в блок управления и выполняются в порядке возрастания их адресов. После выполнения последней команды в ячейки 31—41 приняты на хранение следующие числа:

Адрес	Содержание
31	ce
32	fb
33	af
34	cd
35	ae
36	bd
37	$ce - fb$
38	$af - cd$
39	$ae - bd$
40	$\frac{ce - fb}{ae - bd}$
41	$\frac{af - cd}{ae - bd}$,

представляющие собой промежуточные результаты вычислений, и окончательный результат (в ячейках 40, 41).

Пример 2. Пусть требуется найти решения n заданных систем уравнений:

$$\begin{cases} a_i x + b_i y = c_i \\ d_i x + e_i y = f_i \end{cases} \qquad i = 1, 2, \ldots n.$$

Разрешающий алгоритм представляет собою n-кратное повторение алгоритма решения одной системы такого вида, и соответствующую программу для машины можно было бы легко составить по указанному выше образцу так, чтобы $6n$ коэффициентов располагались в $6n$ ячейках памяти, и чтобы программа насчитывала $11n+1$ команд. После того, как первый цикл из 11 команд выработает решение первой системы, второй цикл из 11 команд выработает решение второй системы и так дальше... n раз, $(11n+1)$-й командой является команда остановки.

Однако такое значительное увеличение объема программы нецелесообразно и его можно избежать. Для этого заметим, что каждый последующий цикл из 11 команд может быть получен из предыдущего путем изменения адресов, участвующих в командах. Именно, если $6n$ коэффициентов расположены по одному в ячейках, следующих подряд одна за другой, например, начиная с № 51, то в первых шести командах адреса множителей должны быть увеличены каждый на 6, для того чтобы получить соответствующие команды второго цикла.

Далее, для того чтобы решение очередной системы оказалось записанным в двух ячейках, следующих за ячейками, хранящими решение предыдущей системы, необходимо увеличить каждый из последних адресов в десятой и одиннадцатой командах на два. Такое изменение адресов может быть осуществлено посредством так называемых *команд переадресации*, которых в нашем случае будет 8. Для этого поместим в ячейки 25 и 26 параметры:

№ ячейки	Содержание
25	0 06 06 00
26	0 00 00 02,

а в ячейки 12—18 восемь команд переадресации:

№ ячейки	Содержание
12	1 01 25 01
13	1 02 25 02
14	1 02 25 03
15	1 04 25 04
16	1 05 25 05
17	1 06 25 06
18	1 10 26 10
19	1 11 26 11

После выполнения команд из ячеек 1—19 в ячейках 40—41, как и прежде, будет принято на хранение решение первой системы уравнений, но вместе с тем в ячейках 1—6 и 10—11 уже окажутся следующие переадресованные команды:

№ ячейки	Содержание
1	3 59 61 31
2	3 62 58 32
3	3 57 62 33
4	3 59 60 34
5	3 57 61 35
6	3 58 60 36
10	4 37 39 42
11	4 38 39 43

Поэтому, если теперь вторично принять к исполнению команды из ячеек 1—19, то в результате такого второго цикла работы машины уже будет найдено решение второй системы уравнений, которое будет направлено на хранение в ячейки 42—43; кроме того, будет осуществлена очередная переадресация команд 1—6 и 10—11, создающая условия для аналогичного третьего цикла работы и т. д.

Как же обеспечить подобное циклическое выполнение 19 команд столько раз, сколько дано систем уравнений с тем, однако, чтобы после того, как будут найдены решения всех заданных систем, машина остановилась? Для этого поместим еще в ячейки 27 и 28 параметры 0 00 00 01 и n (n равно числу всех заданных систем), а к рассмотренным ранее 19 командам присоединим следующие три:

№ ячейки	Содержание
20	2 28 27 28
21	5 01 28 01
22	0 00 00 00

Команда 20 уменьшает на единицу содержимое ячйки 28 после каждого выполнения цикла команд 1—19. Команда 21 — это условная передача управления команде 1: она выполняется до тех пор, пока в ячейке 28 все еще хранится положительное число, и тем самым обеспечивает переход к следующему циклу команд 1—19 для решения следующей системы уравнений. Если же в ячейке 28 хранится 0, а это

случится в точности после n циклов 1—19, то передача управления не осуществляется, а выполняется следующая команда 22, которая останавливает машину.

Из всего сказанного ясно, что для поставленной задачи о решении n систем уравнений пригодна программа из указанных выше команд 1—22, с учетом параметров, вводимых в ячейки 25—28. Структуру этой программы в целом удобно изобразить в виде следующей схемы:

	Команды	Параметры	
1	*Арифметических операций*	25	0 06 06 00
2		26	0 00 00 02
3		27	0 00 00 01
4		28	n
5			
6			
7			
8			
9			
10			
11			
12	*Переадресации*		
13			
14			
15			
16			
17			
18			
19			
20	*Счет циклов*		
21	*Условия передачи управления*		
22	*Остановка*		

Пример 3. Рассмотрим теперь программу для нахождения общего наибольшего делителя двух чисел a, b. Условимся отвести ячейки 12 и 13 для исходных данных a, b соответственно, а ячейки 14 и 15 для промежуточных вычислений с тем, чтобы окончательный результат после остановки машины оказался в ячейке 15. Предлагаемая ниже программа составлена в соответствии с приведенной в § 1 формулировкой алгоритма Евклида.

№ ячейки	Содержание (команда)	Пояснение
01	1 12 05 15	Посылка числа из 12 в 15
02	2 12 13 14	Разность чисел из 12 и 13 отправляется в 14
03	5 02 14 06	Переход к команде 06 при условии, что в 14 отрицательное число
04	5 01 14 09	Переход к команде 09 при условии, что в 14 положительное число
05	0 00 00 00	Остановка
06	1 13 05 12	Посылка числа из 13 в 12
07	1 15 05 13	Посылка числа из 15 в 13
08	5 00 00 01	Безусловный переход к команде 01
09	1 13 05 12	Посылка числа из 13 в 12
10	1 14 05 13	Посылка числа из 14 в 13
11	5 00 00 01	Безусловный переход к команде 01

После первых двух тактов в ячейках 12—15 возникают следующие числа:

№ ячейки	Содержание
12	a
13	b
14	$a-b$
15	a

Если $a-b=0$ (т. е. $a=b$), то команды 03 и 04 об условной передаче управления пропускаются и выполняется команда 05, которая останавливает машину. К этому моменту в ячейке 15, действительно, уже содержится нужный результат (сравнить с указанием 3 § 1).

Если $a-b<0$ (т. е. $a<b$), то команда 03 передает управление команде 06, которая вместе со следующей за ней командой 07 осуществляет перестановку местами чисел a, b в ячейках 12 и 13 (сравнить с указанием 4). Потом команда 08 обеспечивает безусловный переход к 01 и тем самым начинается второй цикл работы машины.

Если же $a-b>0$, т. е. $a>b$, то команда 03 пропускается, а команда 04 передает управление команде 09, которая вместе со следующей за ней командой 10 пересылает в ячейки 12 и 13 прежнее вычитаемое и остаток, т. е. b и $a-b$ соответственно (сравни с указанием 4). Потом команда 11 обеспечивает безусловный переход к команде 01 и тем самым начинается второй цикл работы машины:

Последовательность циклов работы машины будет порождать в ячейках 12 и 13 последовательность пар чисел:

$$(a_1,\ b_1),\ (a_2,\ b_2),\ \ldots,\ (a_i,\ b_i),\ (a_{i+1},\ b_{i+1}),\ \ldots$$

а в ячейке 15 последовательность чисел

$$a_1,\ a_2,\ a_3,\ \ldots\ a_i,\ a_{i+1},\ \ldots$$

до тех пор, пока появится первая пара равных чисел a_k, b_k. Тогда команда 05 остановит машину, а в ячейке 15 будет помещен искомый результат a_k.

Уже в разобранных примерах с достаточной ясностью отражены следующие два основных принципа работы автоматической вычислительной машины:

1. Обычно команды программы выполняются машиной в том порядке, в каком они записаны в ячейках памяти.

Однако машина способна автоматически изменить ход вычислительного процесса в зависимости от получающихся текущих результатов вычислений. Это достигается путем введения условных команд.

2. При сравнительно небольшом объеме программы машина способна производить довольно длинные цепи вычислений благодаря тому, что она может сама преобразовывать и многократно повторять всю программу или отдельные части ее. Такая возможность обеспечивается тем, что программа, закодированная числами, хранится в том же запоминающем устройстве, где хранятся и обычные числа, а следовательно, машина может производить операции и над условными числами, являющимися кодами команд. Так, например, машина может осуществить переадресацию некоторых команд.

Эти же принципы характеризуют работу машины и при решении таких задач, которые не носят явного вычислительного (арифметического) характера. Например, можно запрограммировать алгоритм Тезея (поиск в лабиринте) или известные алгоритмы для проблемы слов в некоторых ассоциативных исчислениях и осуществлять соответствующие процессы в машине. Правда, для этого необходимо, чтобы машина могла производить некоторые дополнительные элементарные операции, кроме арифметических операций, и передачи управления, которые участвовали в рассмотренных выше арифметических задачах. В действующих электронных машинах эти немногие виды простых операций выполнимы (предусмотрены соответствующие команды). Поэтому достаточно только менять программу для того, чтобы та же самая машина переключалась на нужный вид работы.

Однако не только в математике, но и в самых разнообразных других областях человеческой деятельности встречаются процессы, протекающие по строго определенному формальному предписанию (т. е. алгоритму), которые также можно запрограммировать. Так, например, в бухгалтерском и плановом деле анализ поступающих данных, их обработка и составление материальных балансов для получения оптимальных результатов складываются из длинной цепи элементарных операций нескольких типов, которые осуществляются в строгом соответствии со специальными инструкциями и схемами. В других случаях хотя и отсутствует пока еще четкий, завершенный алгоритм, однако его можно было бы создать и довести до формального совершенства. Это относится, в частности, к задаче перевода с одного языка на другой. При достаточной формальной обработке и классификации основных грамматических и стилистических правил, а также приемов пользования словарем можно создать вполне удовлетворительный алгоритм для перевода, скажем, научного или делового текста (для некоторых языков такие алгоритмы уже созданы).

В связи с этим интересно напомнить еще о том, что во многих играх правилам ведения игры присущи определенные черты формального предписания к действию; это обстоятельство позволяет ставить вопрос о разработке алгоритмов для ведения успешной игры. Такой алгоритм, основанный на некоторой тактике игры, и должен указать игроку для каждой позиции единственный (наилучший в смысле этой тактики) ход, который он должен сделать. Так, например, в шахматной

игре можно ввести систему оценок для фигур, при которой король оценивается очень большим числом очков, ферзь меньшим, ладья еще меньшим и т. д., причем пешка имеет наименьшую стоимость. Кроме того, определенным образом оцениваются позиционные преимущества (расположение фигур на доске ближе к центру, подвижность и т. д.). Разность между суммой очков для белых фигур и суммой очков для черных фигур характеризует в смысле данной тактики материальный и позиционный перевес белых над черными на данной стадии игры. Самый простой алгоритм игры заключается в переборке всех ходов, возможных при данной позиции, и в выборе среди них того, который приводит к наибольшему перевесу с точки зрения установленной системы оценок. Лучше, но сложнее будет алгоритм, перебирающий все возможные комбинации из трех или, скажем, пяти ближайших ходов, и на основе этого делающий выбор оптимального хода *).

Из всего сказанного становится ясно, насколько многообразны те виды умственного труда, которые совершаются или могут совершаться в соответствии с определенными алгоритмами. Во всех таких случаях можно в принципе запрограммировать эти алгоритмы и возложить выполнение соответствующей работы на машину с автоматическим управлением. В частности, уже теперь составлены программы для перевода с одного языка на другой и для игры в шахматы, которые успешно выполняются действующими машинами, например машиной БЭСМ Академии наук СССР.

§ 6. НЕОБХОДИМОСТЬ УТОЧНЕНИЯ ПОНЯТИЯ АЛГОРИТМА

Из сказанного выше видна та глубокая связь, которая существует между алгоритмами и автоматическими вычислительными машинами.

*) Более того, теоретически существует наилучший алгоритм, приводящий всегда к выигрышу, когда это только возможно. Для шахматных задач типа *мат черным в N ходов* этот алгоритм предписывает белым следующее: перебирать всевозможные серии из N ходов $Б_1, Ч_2, Б_3, Ч_4, \ldots, Ч_{n-1}, Б_n$, где $Б_1, Б_3, \ldots$ — ходы белых, а $Ч_2, Ч_4, \ldots$ — ходы черных, и выбрать для себя такую последовательность ходов $Б_1, Б_3, \ldots$, которая независимо от того, какими окажутся $Ч_2, Ч_4, \ldots$, приводит к выигрышу. Однако этот алгоритм является настолько громоздким, что его реализация даже на быстродействующих вычислительных машинах практически не осуществима.

Очевидно, всякий процесс, отдельные шаги которого последовательно осуществляются автоматически действующей машиной, может быть описан алгоритмом. С другой стороны, все известные до сих пор алгоритмы, а также все те, которые можно предвидеть при современном состоянии науки, в принципе реализуемы в автоматических машинах.

Последнее утверждение требует некоторого пояснения. Как уже отмечалось, процесс применения алгоритма при решении тех или других задач данного типа может быть сколь угодно длинным, да и записи тех сведений, которыми алгоритм оперирует, могут быть чрезвычайно громоздкими. С другой стороны, запоминающее устройство (память) у современных машин имеет ограниченный объем (так как число ячеек конечно и вместимость каждой ячейки ограничена). Поэтому алгоритм может оказаться при известных условиях практически не осуществимым.

Это можно проиллюстрировать на примере алгоритма Евклида. Простейшая задача нахождения общего наибольшего делителя двух чисел может оказаться неосуществимой для вычислителя, решающего ее «вручную», если для решения этой задачи потребуется больше бумаги и чернил, чем имеется в распоряжении вычислителя. Аналогично этому алгоритм Евклида в данной конкретной задаче может оказаться неосуществимым для машины, если для его осуществления потребуется больший объем памяти, чем тот, которым располагает машина.

Как уже говорилось, в подобных случаях процесс применения алгоритма рассматривается как *потенциально осуществимый процесс*, ведущий после конечного (хотя бы и очень большого) числа шагов к искомому результату.

Говоря же о возможности осуществления алгоритма в машине, имеют в виду потенциальную возможность неограниченного увеличения объема памяти в машине.

Отмеченная связь между понятием алгоритма и понятием автоматической машины с потенциально неограниченной памятью позволяет лучше уяснить сущность каждого из них. Любое уточнение одного из этих понятий представляет одновременно и уточнение другого.

Однако при всем подчеркивании общности этих понятий ни одно из них еще не было нами пока точно определено. Точное математическое определение понятия алгоритма

а вместе с тем и точное определение родственного понятия автоматической вычислительной машины) было выработано в науке лишь в тридцатых годах нашего века. Почему же в течение долгих веков математики обходились без особого беспокойства расплывчатым понятием алгоритма? Почему лишь в сравнительно недавнее время возникла острая потребность в выработке такого строгого математического определения этого понятия, которое могло бы служить предметом математического исследования?

До последнего времени термин алгоритм встречался в математике лишь в связи с построением (созданием) конкретных алгоритмов, когда *утверждение о существовании алгоритма для задач данного типа сопровождалось фактическим его описанием.* При таких обстоятельствах лицу, которому сообщалась система формальных правил, оставалось лишь принимать их к сведению и в процессе их применения убеждаться, что они автоматически приводят к нужному результату. Поэтому вопрос о строгом определении понятия «алгоритм», собственно, и не возникал, и можно было бы довольствоваться расплывчатым, но вместе с тем достаточно близким и понятным каждому математику представлением об алгоритме. Однако по ходу развития математики стали накапливаться такие факты, которые коренным образом изменили эту ситуацию. Причиной этому послужило естественное стремление математиков создавать все более и более мощные алгоритмы, решающие по возможности более широкие классы задач (задачи весьма общего типа). К рассмотрению таких фактов мы сейчас и перейдем.

Вспомним алгоритм для извлечения корня квадратного, который описывается во всех школьных учебниках. Можно поставить более общую цель: построить общий алгоритм для извлечения корня любой степени из любого заданного числа. Естественно ожидать, что такой алгоритм труднее построить, но перспектива обладания таким алгоритмом является привлекательной. Однако можно идти еще дальше. Извлечь корень n-й степени из числа a значит решить уравнение $x^n - a = 0$ (найти корень уравнения). Поэтому можно сформулировать еще более общую задачу.

Построить алгоритм, позволяющий для любого уравнения вида

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0 \qquad (*)$$

(n — произвольное натуральное число *найти все его корни* *).

Конечно, построить такой алгоритм еще труднее; по существу, основное содержание раздела высшей алгебры, именуемого алгеброй многочленов, и сводится к построению и обоснованию этого алгоритма, но вместе с тем и значение его весьма внушительно.

Уже приведенные примеры в какой-то мере характеризуют естественное стремление математиков создавать все более и более мощные алгоритмы, решающие по возможности все более и более широкие классы задач (задачи весьма общего типа). Разумеется, в этом отношении задача о решении любого уравнения вида (*) еще не представляет собою предела, дальше которого нельзя идти. Более того, если мы хотим быть последовательными в своем безудержном стремлении расширять класс задач, для которых желательно иметь единый разрешающий алгоритм, то мы неизбежно должны прийти к постановке следующей проблемы:

Построить алгоритм, позволяющий решать любую математическую задачу.

Такая постановка вопроса является уже столь общей, что поистине может быть расценена как дерзкий вызов всей математике; кроме того, к приведенной формулировке можно было бы придраться хотя бы и потому, что не совсем ясно, что значит «любая математическая задача». Вместе с тем большая притягательная сила подобной проблемы, ее заманчивость не нуждаются в особом рекламировании.

Эта проблема имеет свою историю. Еще великий немецкий математик и философ Лейбниц (1646—1716 гг.) мечтал о создании всеобщего метода, позволяющего эффективно решать любую задачу. Хотя такого всеобщего алгоритма ему и не удалось найти, Лейбниц все же считал, что наступит время, когда такой алгоритм будет найден и тогда любой спор между математиками будет решаться автоматически, с карандашом и бумагой, в соответствии с этим всеобщим алгоритмом.

---

*) Точнее, для любого натурального числа k найти десятичные приближения корней по недостатку и по избытку с точностью до $\frac{1}{10^k}$.

В дальнейшем сама проблема получила определенное уточнение в виде одной из важнейших проблем математической логики, а именно *проблемы распознавания выводимости*. Не имея возможности приводить в этой книжке точного и исчерпывающего изложения вопроса, ограничимся описанием в самых общих чертах.

Как известно, аксиоматический метод в математике заключается в том, что все предложения (теоремы) данной теории получаются посредством формально-логического вывода из нескольких предложений (аксиом), принимаемых в данной теории без доказательства. Раньше всех других математических теорий была осуществлена аксиоматизация геометрии, однако в современной математике почти все математические теории строятся на аксиоматической основе. В математической логике описывается специальный язык формул, позволяющий любое предложение математической теории записать в виде вполне определенной формулы.

Пользуясь терминологией, введенной ранее при рассмотрении ассоциативных исчислений, можно сказать, что всякая такая формула представляет собою слово в некотором специальном алфавите, содержащем наряду с употребительными в математике знаками вроде букв, скобок и т. д., еще и другие специальные знаки, изображающие логические операции, например, отрицание, логическую сумму и т. д. Однако главная аналогия с ассоциативными исчислениями заключается в том, что сам процесс логического вывода из какой-нибудь посылки R следствия S может быть описан в виде процесса формальных преобразований слов, очень схожих с допустимыми подстановками в ассоциативном исчислении. Это позволяет говорить о логическом исчислении, в котором указана система допустимых преобразований, изображающих элементарные акты логического умозаключения, из которых складывается любой сколь угодно сложный формально-логический вывод. Примером такого допустимого преобразования является вычеркивание в формуле двух рядом стоящих знаков отрицания; скажем, «ненекрасивый» может быть преобразовано в «красивый» (интересно сравнить это допустимое преобразование с подстановкой $aa \to \wedge$ в ассоциативном исчислении примера 4).

Вопрос о логической выводимости предложения S из посылки R в указанном логическом исчислении становится

вопросом о существовании дедуктивной цепочки, ведущей от слова, изображающего посылку R, к слову, изображающему предложение S. Проблему распознавания выводимости можно теперь сформулировать так:

для любых двух слов (формул) R и S в логическом исчислении узнать, существует ли дедуктивная цепочка, ведущая от R к S, или нет.

Решение понимается в смысле алгоритма, дающего ответ на любой из вопросов такого типа (любые R и S).

Нетрудно теперь сообразить, что создание такого алгоритма позволило бы получить общий разрешающий метод для автоматического решения самых разнообразных задач из всех математических теорий, которые построены аксиоматически. Действительно, справедливость какого-либо утверждения S (например, формулировка какой-нибудь теоремы) в такой теории понимается как его логическая выводимость из системы аксиом, взятой в качестве посылки R. Но тогда, применяя алгоритм распознавания выводимости, можно было бы установить, справедливо ли это утверждение S в рассматриваемой теории или нет. Более того, в случае утвердительного ответа можно было бы эффективно найти в логическом исчислении соответствующую дедуктивную цепочку, а по ней восстановить цепь умозаключений, составляющих доказательство рассматриваемого утверждения. Фактически, предполагаемый алгоритм решил бы единым эффективным методом почти все сформулированные и еще не решенные до сегодняшнего дня математические проблемы. Это обстоятельство делает понятным не только заманчивость построения такого «всеобщего алгоритма», а значит и построения соответствующей «всемогущей машины», но и трудности такого построения.

И действительно, несмотря на долгие и упорные усилия многих крупных специалистов, трудности, связанные с таким построением, остались непреодоленными. Более того, подобные трудности были вскоре обнаружены уже при попытке создания алгоритмов для некоторых математических задач гораздо более частного типа; к ним, в частности, относится и проблема Гильберта о диофантовых уравнениях (см. § 1), а также ряд других задач, о которых будет сообщено ниже.

В результате многочисленных, но безрезультатных попыток создания таких алгоритмов стало ясным, что налицо имеются трудности принципиального характера, и возникло

подозрение, что *далеко не для всякого класса задач возможно построение разрешающего алгоритма.*

Очевидно, утверждение об алгоритмической неразрешимости некоторого класса задач, т. е. о невозможности указать соответствующий разрешающий алгоритм, является не просто признанием того, что такой алгоритм нам не известен и никем еще не найден. Такое утверждение представляет собою одновременно и прогноз на все будущие времена о том, что алгоритм никогда и никем не будет указан (иными словами, что он не существует), и оно нуждается в обосновании посредством какого-нибудь математического доказательства. Однако о подобном доказательстве и думать нет смысла, пока отсутствует точное определение понятия «алгоритм», ибо в противном случае не ясно, несуществование чего мы собираемся доказывать. Полезно здесь напомнить, что в истории математической науки были известны и ранее такие проблемы, которые долгое время не поддавались решению и относительно которых впоследствии было установлено, что они не разрешимы теми средствами, которыми ранее пытались их решать. В качестве примеров укажем на проблемы трисекции угла и решения уравнений в радикалах.

Из школьного курса известно, как осуществляется биссекция угла при помощи циркуля и линейки. Еще у древних греков возникла аналогичная задача о трисекции угла при помощи циркуля и линейки. Однако было доказано, что трисекцию произвольного угла невозможно осуществить этими средствами.

Из школьного курса также известно, что корни квадратного уравнения выражаются через его коэффициенты при помощи формулы, в которой фигурируют знаки арифметических операций и знак радикала (второй степени). Для уравнений третьей и четвертой степеней также установлены формулы в радикалах, которые, правда, значительно сложнее, причем в них встречаются «многоэтажные» радикалы. Однако поиски подобных формул в радикалах для уравнений степени выше четвертой, продолжавшиеся до начала XIX века, оказались безуспешными, пока, наконец, был установлен следующий замечательный результат:

Ни для какого n, большего или равного пяти, невозможно указать формулу, которая выражала бы корни любого уравнения n-й степени через его коэффициенты при помощи радикалов.

В обоих этих случаях доказательства невозможности сами оказались возможными, поскольку имелись два точных определения, дающих ответы на вопросы: «что значит построение с помощью циркуля и линейки?» и «что значит решение уравнения в радикалах?». Заметим кстати, что в этих двух определениях уточняется смысл некоторых алгоритмов специального вида, а именно алгоритма решения уравнений в радикалах (а не вообще алгоритма для решения уравнений) и алгоритма трисекции любого угла при помощи циркуля и линейки (а не вообще алгоритма трисекции). Между тем, для общего понятия *алгоритм* мы точного определения до недавнего времени не имели, и поэтому выработка такого определения стала одной из важных задач современной математики. Весьма важно подчеркнуть, что выработку определения понятия алгоритм (как и, впрочем, всякого другого математического определения) нельзя рассматривать как произвольное соглашение математиков в одинаковом понимании термина *алгоритм*. При формулировке этого определения пришлось преодолеть трудности, заключающиеся в том, что предлагаемое определение должно правильно отражать сущность того понятия, которое фактически уже имелось, хотя и в расплывчатом виде, и которое иллюстрировалось нами на многочисленных примерах. С этой целью, начиная с тридцатых годов XX века, был предпринят ряд исследований для выявления всех тех средств, которые фактически привлекаются для построения алгоритмов. Задача состояла в том, чтобы на этой основе дать определение понятия *алгоритм*, которое было бы совершенным не только с точки зрения формальной точности, но и, главное, с точки зрения фактического соответствия сущности определяемого понятия. При этом различные исследователи исходили из разных технических и логических соображений, и вследствие этого было выработано несколько определений понятия *алгоритм*. Однако впоследствии выяснилось, что все эти определения равносильны между собой и, следовательно, определяют одно и то же понятие; это и есть современное точное понятие *алгоритм*.

То обстоятельство, что все способы уточнения понятия *алгоритм*, несмотря на свое различие и многообразие, всегда приводили и приводят по существу к одному и тому же результату, имеет большое познавательное значение; оно свидетельствует как раз об удачности выработанного определения.

С точки зрения машинной математики особый интерес представляет то определение алгоритма, в котором сущность этого понятия раскрывается путем рассмотрения процессов, осуществимых в машине.

Для такого строгого математического определения нужно изобразить механизм работы машины в виде некоторой стандартной схемы, максимально простой по своей логической структуре, но вместе с тем настолько точной, чтобы она могла служить предметом математического исследования. Впервые это было сделано английским математиком Тьюрингом, который предложил самую общую и вместе с тем самую простую концепцию вычислительной машины. Следует отметить, что машина Тьюринга была описана в 1937 году, т. е. до создания современных счетных машин, При этом Тьюринг исходил лишь из общей идеи уподобления работы машины работе вычислителя, оперирующего в соответствии с некоторым строгим предписанием. Предлагаемое же нами изложение этого вопроса использует уже общие принципы работы существующих электронных вычислительных машин.

§ 7. МАШИНА ТЬЮРИНГА

Отличительные особенности машины Тьюринга по сравнению с электронными машинами, описанными в §§ 4—5, заключаются в следующем:

1. В машине Тьюринга расчленение процесса на простые элементарные операции доведено в известном смысле до предельной возможности. Так, например, операция сложения, фигурирующая в электронной машине как единичная элементарная операция, здесь сама расчленяется на цепочку еще более простых операций.

Разумеется, это значительно удлиняет процесс, реализуемый в машине Тьюринга, но вместе с тем логическая структура процесса сильно упрощается и приобретает весьма удобный для теоретических исследований стандартный вид.

2. В машине Тьюринга часть памяти *) изображается в виде неограниченной с обеих сторон ленты, разбитой на ячейки.

Очевидно, ни в одной реально осуществимой машине не может быть бесконечной памяти (бесконечной ленты), и в этом смысле машина Тьюринга представляет собою

*) А именно, так называемая внешняя память.

лишь идеализированную схему, отображающую потенциальную возможность увеличения объема памяти.

Такая идеализация оправдывается ранее уже отмечавшейся связью между понятием алгоритма и понятием машины с потенциально неограниченной памятью.

Перейдем теперь к подробному описанию машины Тьюринга.

1. Машина располагает конечным числом знаков (символов)

$$s_1, s_2, \ldots, s_k,$$

образующих так называемый *внешний алфавит*, в котором кодируются сведения, подаваемые в машину, а также те, которые вырабатываются в ней. Для общности последующих рассмотрений удобно принять, что среди знаков внешнего алфавита имеется *пустой знак* (пусть это будет для определенности s_1), посылка которого (вписывание которого) в какую-либо ячейку ленты (памяти) гасит (стирает) тот знак, который в ней раньше хранился, и оставляет ее пустой. О пустой ячейке будем говорить, что она хранит пустой знак.

На любой стадии работы машины в каждой ячейке может храниться не более одного знака. Каждое сведение, хранящееся на ленте, изображается конечным набором знаков внешнего алфавита, отличных от пустого знака и хранящихся по одному в некоторых ячейках ленты. К началу работы машины на ленту подается начальное сведение (начальная информация); работа машины складывается из следующих один за другим тактов, по ходу которых происходит преобразование начальной информации в промежуточные информации (к концу каждого такта совокупность знаков, хранящихся на ленте, образует соответствующую промежуточную информацию). В качестве начальной информации на ленту можно подать любую конечную систему знаков внешнего алфавита (любое слово в этом алфавите), расставленную произвольным образом по ячейкам. Однако в зависимости от того, какая была подана начальная информация $\mathfrak{A}$, возможны два случая:

а) после конечного числа тактов машина останавливается, подавая сигнал об остановке; при этом на ленте оказывается изображенной некоторая информация $\mathfrak{B}$. В таком случае говорят, что машина применима к начальной информации $\mathfrak{A}$ и перерабатывает ее в результирующую информацию $\mathfrak{B}$;

б) остановка и сигнал об остановке никогда не наступают. В таком случае говорят, что машина не применима к начальной информации $\mathfrak{A}$.

Говорят, что *машина решает некоторый класс задач*, если машина применима всегда к информации, изображающей в определенном коде условия любой отдельной задачи этого типа и перерабатывает ее в информацию, изображающую в том же коде решение этой задачи.

2. Система трехадресных приказов, принятая во многих из действующих электронных машин, обусловлена наличием элементарных операций, в которых участвует сразу содержание трех ячеек памяти.

Однако в некоторых электронных машинах использована система одноадресных приказов, связанная с тем, что в каждом такте участвует лишь одна ячейка памяти. (Эту ячейку будем называть *обозреваемой ячейкой* на данном этапе.) Так, например, трехадресный приказ о сложении чисел из ячеек β, γ с отправкой результата в ячейку δ может быть заменен при надлежащих условиях тремя последовательными приказами: *а*) вызов (в сумматор) числа из ячейки β, *б*) вызов числа из ячейки γ, *в*) отправка результата в ячейку δ.

В машине Тьюринга система элементарных операций и вместе с нею система одноадресных приказов еще больше упрощены: на каждом отдельном такте приказ предписывает лишь замену единственного знака s_i, хранящегося в обозреваемой ячейке, каким-либо другим знаком s_j. При $j = i$ это означает, что содержание обозреваемой ячейки не изменяется; при $j = 1$ это означает, что если в обозреваемой ячейке хранился какой-либо знак, то он гасится. Дальнейшее упрощение заключается в том, что при переходе машины от одного такта к непосредственно следующему такту адрес обозреваемой ячейки может изменяться не более, чем на одну единицу, т. е. обозревается соседняя слева или соседняя справа ячейка, или же обозревается та же ячейка, что и в предыдущем такте.

Идея этого упрощения заключается в том, что необходимое для процесса содержание какой-либо отдельной ячейки отыскивается путем постепенной проверки всех ячеек подряд до тех пор, пока не будет обнаружена нужная ячейка. Разумеется, это очень удлиняет процесс, но вместе с тем возникает следующее удобство: в командах программы вместо произвольных адресов обозреваемых ячеек можно ограни-

чиваться применением всего лишь трех стандартных адресов, которые изображаются специальными знаками:

$П$ — обозревать соседнюю справа ячейку,

$Л$ — обозревать соседнюю слева ячейку,

$Н$ — продолжать обозревать ту же ячейку, что и прежде.

3. Для обработки числовой информации, хранящейся в памяти, электронная машина, описанная в §§ 4—5, имеет арифметический блок $\mathscr{L}$, который может пребывать в одном из конечного числа состояний: складывающее, вычитающее и т. д.

Для осуществления какой-либо операции в блок подаются по определенным каналам не только числа, над которыми операции совершаются, но также сигналы, настраивающие блок на соответствующую операцию, т. е. на соответствующее состояние (см. рис. 4,*б*). В машине Тьюринга обработка информации происходит *в логическом блоке*, который также может пребывать в одном из конечного числа состояний; пусть

$$q_1, q_2, \ldots, q_m$$

— специальные знаки, введенные для обозначения этих состояний. Блок имеет два входных канала; по одному из них на каждой стадии работы машины (в каждом такте) поступает знак из обозреваемой ячейки, по другому — знак q_l того состояния, которое предписывается блоку на данный такт; по выходному же каналу блок посылает в обозреваемую ячейку соответствующий, «переработанный» знак s_j, являющийся однозначной функцией от сигналов s_j, q_l, поданных на вход. Команды, обусловливающие срабатывание машины на каждом отдельном такте, имеют вид:

$$Пq_l,\ Лq_l,\ Hq_l \quad (l = 1, 2, \ldots, m),$$

где первый знак заменяет адрес обозреваемой ячейки (см. выше), а второй предписывает логическому блоку надлежащее состояние. Знаки $П$, $Л$, $Н$, $q_1, \ldots, q_m$ образуют *внутренний алфавит* машины.

Специфической особенностью машины Тьюринга является то, что на логический блок возлагается также и выработка на каждом данном такте той команды, которая будет подаваться в блок управления к началу следующего такта. Таким образом, логический блок имеет, кроме канала для выдачи знака s_j, еще два канала для выдачи двух знаков очередной команды. Соответствующая схема дана на рис. 5.

При этом существенным является то, что *выходная тройка* знаков s_j, P, q_l*) зависит исключительно от того, какая *входная пара* знаков s_i, q_n была подана в том же такте на входы блока. Это означает, что логический блок реализует функцию, сопоставляющую каждой паре знаков s_i, q_n (всего таких пар $k \cdot m$) тройку знаков s_j, P, q_l. Эту функцию, которую мы будем называть *логической функцией* машины, удобно изобразить в виде прямоугольной таблицы, столбцы которой занумерованы знаками состояний, а строки — знаками внешнего алфавита; в каждой же ячейке таблицы записана соответствующая выходная тройка знаков. Эту таблицу будем называть *функциональной схемой* машины; на рис. 6 приводится пример такой схемы.

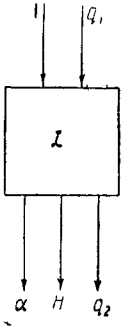


Рис. 5.

Из приведенного описания ясно, что *работа* машины Тьюринга вполне определяется той логической функцией, которую реализует логический блок. Иными словами, две машины Тьюринга с общей функциональной схемой неразличимы, коль скоро мы интересуемся лишь тем, как они работают. С другой стороны, структура машины, состав отдельных ее органов и их взаимодействие можно задать в виде *структурной схемы*, общей для всех машин Тьюринга (см. рис. 7).

	q_1	q_2	q_3	q_4	q_5
Λ	$\Lambda\Pi q_4$	$\Lambda Л q_3$	$\Lambda\Pi q_1$	$\Lambda H q_5$	$\Lambda H q_5$
1	$\alpha H q_2$	$\beta H q_1$	$1\Pi q_1$	$1Л q_1$	$1H q_5$
α	$\alpha Л q_1$	$\alpha\Pi q_2$	$1Л q_3$	$\Lambda\Pi q_4$	$\alpha H q_5$
β	$\beta Л q_1$	$\beta\Pi q_2$	$\Lambda Л q_3$	$1\Pi q_4$	$\beta H q_5$

Рис. 6.

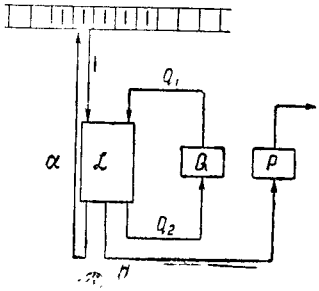


Рис. 7.

В этой схеме отражено разделение памяти на внешнюю и внутреннюю. Внешняя память изображена ячейками бесконечной ленты, предназначенными для хранения информации, закодированной в символах внешнего алфавита; внутренняя память — двумя ячейками для хранения очередной

*) Под P подразумевается з ков $П$, $Л$, $Н$.

команды: Q-ячейка хранит знак состояния и P-ячейка — знак сдвига. В этих двух ячейках происходит задержка знаков P, q_l, полученных на выходе логического блока в данном такте работы, до начала следующего такта, когда они поступают в блок управления. Функции блока управления теперь уже чрезвычайно упрощены и заключаются по существу лишь в обеспечении сдвига ленты не более, чем на одну ячейку, в соответствии с поступившим P-знаком. Знак состояния можно было бы фактически подавать из Q-ячейки прямо в $\mathscr{L}$, образуя так называемую линию обратной связи, по которой в блок $\mathscr{L}$ поступает знак q_l, выработанный в нем же в предыдущем такте.

Работа машины Тьюринга протекает следующим образом. Перед ее запуском на ленту заносится начальная информация (на нашем рис. 7 последовательность из пяти палочек) и в «поле зрения» машины устанавливается определенная начальная ячейка (на чертеже — ячейка, содержащая четвертую справа палочку), а в ячейки P и Q вводятся знаки начального состояния и начального сдвига (предположим q_1 и H). Дальнейший процесс протекает уже автоматически и однозначным образом определяется функциональной схемой машины. Посмотрим, например, что произойдет в том случае, когда задана функциональная схема рис. 6.

Первый такт. Обозревается знак | (палочка) из начальной ячейки (сдвиг H) при состоянии q_1. Результат: выходная тройка знаков αHq_2, т. е. знак | заменен знаком α, а в ячейки P, Q послана на хранение до следующего такта очередная команда Hq_2.

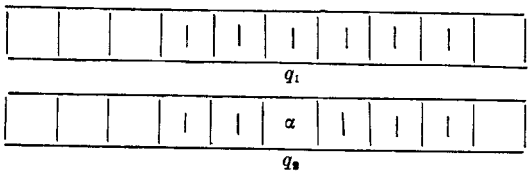
Второй такт. Обозревается знак α из той же ячейки (сдвиг H) при состоянии q_2. Выходная тройка: $\alpha Пq_2$, т. е. знак α оставляется без изменения с переходом к команде $Пq_2$.

Третий такт. Обозревается палочка из соседней справа ячейки (сдвиг $П$), при состоянии q_2. Результат: знак | заменен знаком β с переходом к команде Hq_1 и т. д.

Как видно из последнего столбца функциональной схемы рис. 6, остановка машины произойдет лишь при условии, что на некоторой стадии процесса возникнет состояние q_5. Действительно, каков бы ни был обозреваемый знак, он не будет заменен никаким другим, и машина будет продолжать обозревать его (сдвиг H) при том же состоянии q_5. Это и есть *стоп-состояние*, сигнализирующее о результативном завершении процесса в том случае, когда машина применима к информации, поданной в нее до запуска.

По существу функциональной схемой может пользоваться и человек-вычислитель; она представляет для него некоторый стандартный способ задания алгоритма преобразования исходных данных, записанных во внешнем алфавите, в соответствующий результат, записанный в том же алфавите. Фактически мы так и поступили только что с функциональной схемой рис. 6 при переработке слова из пяти палочек, сознавая при этом, что машина Тьюринга сделала бы то же самое. Впредь в подобных случаях мы для большей наглядности будем пользоваться так называемыми *конфигурациями*. Под k-й *конфигурацией* мы будем понимать изображение ленты машины с информацией, сложившейся на ней к началу k-го такта, причем под обозреваемой ячейкой записан знак того состояния, которое подается в логический блок $\mathscr{L}$ к началу этого же такта. Таким образом в k-й конфигурации явно указана входная пара знаков, а, следовательно, можно, обращаясь к функциональной схеме, определить соответствующую выходную тройку, а тем самым и $(k+1)$-ю конфигурацию.

В разобранном выше примере первая и вторая конфигурации таковы:



со входными парами $1\,q_1$, αq_2 соответственно. Переход от первой конфигурации ко второй обусловливается выходной тройкой $\alpha H q_2$, соответствующей в схеме рис. 6 входной паре $1q_1$.

	q_1	q_2	q_3	q_4
Λ	Πq_4	Πq_3	Πq_4	!
1	αq_2	Πq_3	Πq_1	!
α	Π	Π	$1\,\Pi$	$\Lambda\,\Pi$
β	Π	Π	$\Lambda\,\Pi$	$1\,\Pi$

Рис. 8.

Условимся еще в следующей упрощенной записи функциональных схем, которая делает схему более обозримой и удобной при выписывании конфигураций. Именно, мы откажемся от полной записи выходной тройки $s_j P q_l$, опуская знаки s_j и q_l, если они не отличаются от соответствующих входных знаков, а также знак H, указывающий на отсутствие сдвига. В частности, это позволяет полностью опустить столбец, соответствующий стоп-состоянию.

Упрощённая запись для схемы рис. 6 приведена на рис. 8, на котором стоп-состояние обозначено знаком «!». Из столбца q_1 схемы рис. 8 проще усматривается, чем из рис. 6, что, оказавшись в состоянии q_1 при обозреваемом знаке α, машина начинает серию сдвигов влево сквозь все рядом стоящие знаки α и β, продолжая оставаться в состоянии q_1 и не изменяя содержания обозреваемых ячеек, до тех пор, пока в ее поле зрения попадет впервые палочка или пустая клетка; только при этих условиях машина выйдет из состояния q_1.

Впредь знак «!» будет всюду применяться для обозначения стоп-состояния.

§ 8. РЕАЛИЗАЦИЯ АЛГОРИТМА В МАШИНЕ ТЬЮРИНГА

В этом параграфе на ряде примеров мы покажем, как строятся тьюринговы машины, реализующие некоторые простые арифметические алгоритмы, и как происходит в машине процесс реализации этих алгоритмов; в соответствии с содержанием предыдущего параграфа под построением машины мы будем понимать построение функциональной схемы ее логического блока, которая и представляет собою некоторую стандартную форму записи алгоритма. Кроме того, будут изложены и некоторые более общие соображения о методике построения тьюринговых машин (функциональных схем).

I. Алгоритм перехода от n к $n+1$ в десятичной системе счисления

Решается задача следующего типа:

Дана десятичная запись числа n (т. е. представление натурального числа n в десятичной системе счисления); *требуется указать десятичную запись числа* $n+1$.

Для этого берется внешний алфавит, состоящий из десяти цифр 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 и пустого знака $\wedge$. Машина может пребывать лишь в двух состояниях: q_0 (рабочее состояние) и! (остановка). Заданное число n, а также результирующее число $n+1$ будут записаны в десятичной системе, причем цифры будут помещаться по одной в ячейке (ячейки следуют подряд одна за другой без пропусков). Соответствующая функциональная схема дана на рис. 9 в виде той части указанной на нем таблицы, которая получится, если

не учитывать в ней последнюю строку и последний столбец (смысл расширенной таблицы будет пояснен несколько позже). Пусть к началу работы в поле зрения установлена цифра разряда единиц числа n, а машина настроена в состояние q_0; если эта цифра отлична от 9, то машина остановится уже после первого такта работы, в котором происходит замена этой цифры другой цифрой в соответствии со схемой. Если же последняя цифра 9, то машина заменяет ее нулем и производит сдвиг влево (к соседнему, более высокому разряду) и продолжает пребывать в рабочем состоянии q_0 (таким образом обеспечивается перенос единицы в более высокие разряды). Если число оканчивается k девятками, то машина закончит работу в точности после $k+1$ такта. На рис. 10 выписаны соответствующие конфигурации при $n = 389$.

	q_0	q_1
0	1 !	
1	2 !	
2	3 !	
3	4 !	
4	5 !	
5	6 !	
6	7 !	
7	8 !	
8	9 !	
9	0 Л	
Λ	1 !	
1	Л	Λ $Лq_0$

Рис. 9.

Поясним теперь смысл расширенной таблицы, помещенной на рис. 9. Она задает функциональную схему машины, в которой имеется еще одно состояние — q_1; кроме того, во внешнем алфавите имеется еще один знак, а именно «палочка». Если к началу работы машина настроена в состоянии q_0, а на ленте нет ни одной палочки, то работа будет протекать в точности так же, как если бы мы имели дело с машиной предыдущего примера. Это непосредственно очевидно из того, что при указанных условиях последняя строка и последний столбец таблицы не принимают никакого участия в описании работы. В частности, это означает, что данная машина может также быть использована для реализации предыдущего алгоритма. Однако данная машина способна делать еще кое-что и ради этого мы как раз и стали ее рассматривать.

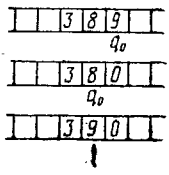


Рис. 10.

Пусть на ленте задана десятичная запись числа n, а в нескольких ячейках, подряд расположенных правее этой записи, записаны палочки, по одной в ячейке. Посмотрим, как будет работать машина с этой функциональной схемой, если к началу работы в поле зрения установлена самая правая палочка, а сама машина настроена в состоянии q_1. В первом

такте (входная пара $q_1|$) стирается эта палочка, а также происходит сдвиг налево и переход в состояние q_0 (выходная тройка $\wedge Лq_0$). В последующих тактах машина продолжает сдвиг налево при состоянии q_0, сквозь все палочки до первой цифры разряда единиц. Начиная с этого момента все протекает уже как и в предыдущем алгоритме, т. е. происходит переработка записи числа n в запись числа $n+1$, и процесс завершается.

Короче, машина уменьшает на единицу число палочек и осуществляет в десятичной записи переход от числа n к числу $n+1$. Условимся этот процесс называть *контро-*

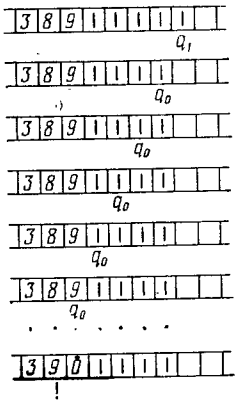


Рис. 11.

	q_0	q_1	q_2
0	1 q_2	!	$П$
1	2 q_2	!	$П$
2	3 q_2	!	$П$
3	4 q_2	!	$П$
4	5 q_2	!	$П$
5	6 q_2	!	$П$
6	7 q_2	!	$П$
7	8 q_2	!	$П$
8	9 q_2	!	$П$
9	0 $Л$	!	$П$
$\wedge$	1 q_2	!	$Лq_1$
!	$Л$	$\wedge Лq_0$	$П$

Рис. 12.

лируемым переходом от десятичной записи n к десятичной записи $n+1$.

На рис. 11 выписаны конфигурации для набора из пяти палочек и для $n=389$.

II. Алгоритм перевода в десятичную систему счисления

Построим функциональную схему машины (алгоритма), решающей задачу следующего типа:

Дана конечная совокупность палочек, вписанных в ячейках, взятых подряд без пропусков (такие совокупности будем называть *наборами палочек*); *требуется записать в десятичной системе число этих палочек.*

Короче: пересчитать набор палочек.

Такая схема задана на рис. 12. Для того, чтобы убедиться в том, что эта схема действительно описывает нужную машину (алгоритм), полезно сравнить ее со схемой расширенной таблицы рис. 9. Столбец q_0 в схеме рис. 12 отличается от столбца q_0 в схеме рис. 9 лишь тем, что вместо состояния «!» в нем всюду фигурирует новое состояние q_2; различие столбцов q_1 не имеет существенного значения в работе схемы рис. 9. Поэтому, если на ленте заданы десятичная запись числа n, а правее ее набор палочек, если, далее в поле зрения машины, как и прежде, установлена самая правая палочка, а сама машина настроена в состоянии q_1, то в машине будет вначале протекать такой же процесс, как и при схеме рис. 9; именно палочка набора будет стерта, а запись числа n будет заменена записью числа $n+1$. Однако в то время как согласно схеме рис. 9 на этой стадии процесса появится состояние !, т. е. процесс прекращается, согласно схеме рис. 12 появляется состояние q_2 и процесс продолжается.

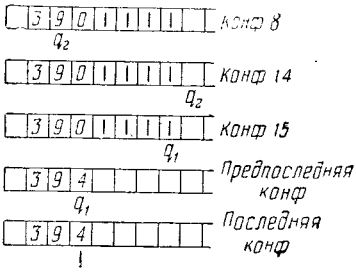


Рис. 13.

В частности, если первая конфигурация взята как на рис. 11, то восьмая конфигурация получится уже такой, как на рис. 13. Как процесс будет продолжаться, видно из столбца состояния q_2: начнется серия сдвигов вправо сквозь все цифры и все палочки, пока будет достигнута первая пустая ячейка при входной паре $\wedge\, q_2$ (см. конфигурацию 14 рис. 13), тогда последует сдвиг налево с одновременным переходом в состояние q_1 (конфигурация 15 рис. 13). Таким образом, в поле зрения машины окажется опять самая правая палочка набора при состоянии q_1, тем самым кончается один цикл работы и начинается второй цикл работы, аналогичный первому. В результате второго цикла будет стерта еще одна палочка, а запись числа $n+1$ будет заменена записью числа $n+2$. Если первоначально в наборе было k палочек, то после k циклов работы все они будут стерты, а вместо первоначальной записи числа n появится запись числа $n+k$. По окончании k-го цикла машина опять окажется в состоянии q_1, но в ее поле зрения уже будет не палочка (палочки все уже стерты), а самая первая цифра (т. е. цифра единиц) в записи числа $n+k$ (предпоследняя конфигурация на рис. 13).

Как видно из схемы рис. 12, в этом случае произойдет остановка (последняя конфигурация на рис. 13).

Из всего сказанного вытекает, что если к началу работы машины на ленте будут записаны цифра 0 и набор из k палочек, то машина сотрет все эти палочки, а вместо нуля появится десятичная запись числа $0+k$, т. е. k. На самом же деле к началу работы можно обойтись и без нуля, так как если вместо нуля фигурирует знак $\wedge$, то при состояниях q_0 и q_1 машина ведет себя так же, как если бы это был нуль (см. схему рис. 12). Итак, предложенная схема рис. 12 действительно описывает алгоритм перевода набора палочек в десятичную запись их числа.

Упражнение. Составить по аналогии с I функциональную схему машины (алгоритма), реализующей переход от десятичной записи числа n к десятичной записи числа $n-1$ (при $n \geqslant 1$). Далее, по аналогии с II составить функциональную схему машины для перевода с десятичной системы счисления, т. е. для замены десятичной записи какого-либо числа n набором из n палочек.

Мы рассмотрим еще некоторые примеры Тьюринговых машин для решения арифметических задач. В этих задачах как исходные данные (условия задачи), так и результат являются натуральными числами. Мы условимся считать, что в машине каждое натуральное число задано в виде набора из такого же числа палочек; если же в задаче фигурируют несколько натуральных чисел, то изображающие их наборы палочек будем отделять каким-нибудь специальным знаком, например звездочкой *. Этот знак тоже входит во внешний алфавит машины.

III. Алгоритм сложения

На ленту подается пара чисел, например,

В результате должна получиться их сумма, в данном случае

Заметим, что работу машины нельзя свести просто к стиранию звездочки, ибо тогда возникает пустая ячейка между

палочками, а, следовательно, совокупность оставшихся палочек нельзя будет рассматривать как изображение одного натурального числа. Согласно предполагаемой функциональной схеме (см. рис. 14) работа машины будет протекать так.

	q_0	q_1	q_2
|	$\wedge П q_2$	$П$	$П$
$\wedge$	$П$	$П q_0$	| q_1
*	$\wedge$!	$П$	$П$

Рис. 14.

Начальные условия: в поле зрения установлена самая левая палочка и машина настроена в состоянии q_0 (конфигурация 1 на рис. 15).

Первый такт. Обозреваемая палочка стирается, сдвиг вправо (в поле зрения устанавливается следующая палочка) и переход к состоянию q_2 (конфигурация 2).

Последующие такты, как видно из столбца q_2, сводятся к сдвигам направо сквозь все палочки и звездочку до тех

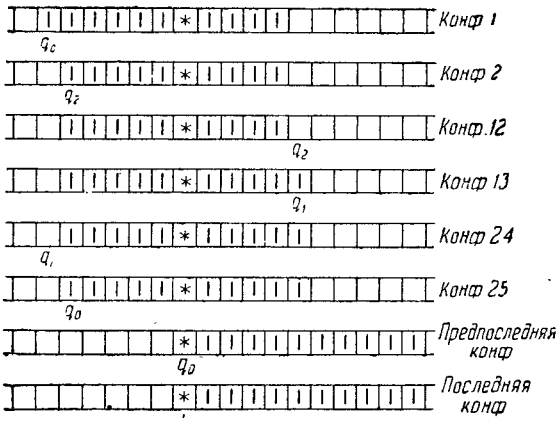


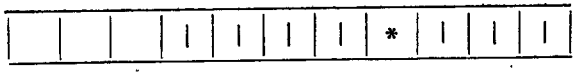
Рис. 15.

пор, пока не будет достигнута первая пустая ячейка (конфигурация 12); тогда (входная пара $\wedge q_2$) в эту пустую ячейку вписывается палочка и машина переходит в состояние q_1 (конфигурация 13). При состоянии q_1 происходят сдвиги влево сквозь все палочки и звездочку до первой пустой ячейки слева (конфигурация 24), тогда (входная пара $\wedge q_1$) происходит сдвиг вправо, в поле зрения устанавливается первая из оставшихся палочек левее звездочки, и машина переходит в состояние q_0 (конфигурация 25). В результате этого цикла одна палочка левого слагаемого оказалась

перенесенной в правое слагаемое. Если левее звездочки было первоначально k палочек, то через k циклов все будут перенесены правее звездочки. При $(k+1)$-м заходе вправо, в поле зрения машины при состоянии q_0, попадает уже не палочка (их нет уже левее звездочки), а сама звездочка (предпоследняя конфигурация). Тогда (входная пара $^*q_0$) звездочка вычеркивается, и машина останавливается (последняя конфигурация). Вместе с тем получена уже и нужная сумма.

IV. Алгоритмы повторного суммирования и умножения

Посмотрим, как нужно изменить схему рис. 14 для того, чтобы при первоначальной подаче на ленту пары чисел m, n, например



машина впадала бы в бесконечный процесс, заключающийся в том, что левое число m прибавляется к правому и потом оно прибавляется к полученной сумме $n+m$, потом к сумме $n+2m$ и т. д. без конца. Для этого нужно, очевидно, чтобы левое слагаемое не исчезло бесследно после первого сложения, а, наоборот, чтобы его можно было восстановить после каждого сложения и снова прибавить к числу, изображенному правее звездочки. Этого можно добиться, например, тем, что палочки левого набора не стираются, а временно заменяются какими-то знаками или пометками. На рис. 16 изображена схема, в которой роль такой пометки играет буква α. В соответствии с этим, вхождение пустого знака $\wedge$ в первой строке схемы 14 отвечает вхождение знака α в первой строке схемы 16 и, кроме того, в схеме рис. 16 имеется еще четвертая строка для буквы α; в этой строке первые три клетки такие же, как в строке для $\wedge$ на рис. 14.

	q_0	q_1	q_2	q_3
|	α Л q_2	Л	Л	‹
$\wedge$	Л	Л q_0	| q_1	Л q_0
*	q_3	Л	Л	Л
α	Л	Л q_0	| q_1	| Л

Рис. 16.

Далее, для того чтобы процесс не прекратился после первого сложения, необходимо, чтобы в схеме рис. 16 вместо знака «!», который в схеме рис. 14 появляется при входной паре $^*q_0$, входил знак другого состояния, обеспе-

чивающего продолжение процесса; в нашем случае таким будет дополнительно введенное состояние q_3. Для этого знака нужно ввести и соответствующий ему столбец.

В схеме рис. 16 нет знака «!» (остановки); поэтому процесс, описываемый ею, не имеет конца. Читатель теперь без особого труда проверит, что соответствующая машина реализует как раз неограниченное прибавление числа, изображенного левее звездочки, к числу, изображенному правее ее. Если правее звездочки к началу работы нет палочек (т. е. второе число нуль), то правее звездочки появятся m палочек, потом $2m$, потом $3m$ и т. д. без конца.

Упражнение. Составить функциональную схему для алгоритма умножения.

Указание. Взять за основу предыдущую схему и изменить ее так, чтобы процесс повторного суммирования не продолжался неограниченно, а выполнялся столько раз, сколько единиц во множителе (после каждого цикла сложения стирается одна палочка из набора, изображающего множитель).

V. Алгоритм Евклида

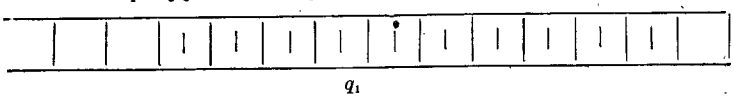
Рассмотрим теперь, как выглядит в машине Тьюринга алгоритм Евклида для нахождения общего наибольшего делителя чисел a, b. Этот алгоритм был уже ранее описан нами дважды: первый раз в виде словесного предписания, а второй раз в виде программы для электронной машины с автоматическим управлением. На этот раз алгоритм мы зададим в виде функциональной схемы Тьюринговой машины и проследим процесс вычисления в машине. Этот процесс складывается из чередующихся попеременно *циклов сравнения* и *циклов вычитания*, соответствующих *элементарным операциям сравнения* и *вычитания* в электронной машине. Соответствующая функциональная схема изображена на рис. 8; ее внешний алфавит состоит из четырех знаков

$$\wedge,\ |,\ \alpha,\ \beta.$$

Натуральные числа будут, по-прежнему изображаться соответствующими наборами палочек. Во избежание таких деталей, которые не связаны с существом дела и способны лишь усложнить наши рассмотрения, условимся наборы палочек, изображающие заданные два числа, располагать на ленте один за другим без пропусков и не отделяя их звез-

дочкой, причем в начале процесса в поле зрения машины установлена самая правая палочка в наборе первого числа. После подробного разбора, который будет приведен ниже, читатель в качестве упражнения, без особого труда, сумеет изменить предложенную функциональную схему так, чтобы она обеспечивала правильную работу машины и при другом способе задания условий задачи (например, если наборы отделены звездочкой, а в поле зрения машины установлена какая-нибудь пустая ячейка). Отметим еще, что буквы α, β будут играть роль временных пометок, которые вычислитель делает (обычно в виде штрихов или галочек) для запоминания некоторых обстоятельств, возникающих по ходу процесса.

Дальнейшее описание будем сопровождать наглядной иллюстрацией применительно к случаю $a = 4$, $b = 6$ посредством конфигурации. Первая конфигурация выглядит так:



В цикле сравнения участвуют лишь состояния q_1, q_2; в цикле вычитания участвуют же q_3 и q_4.

Проследим теперь процесс подробней. Вначале машина сравнивает числа, изображенные на ленте для того, чтобы установить, какое из них больше. При этом она поступает так же, как стал бы делать человек, сравнивающий две длинные последовательности из единиц, которые трудно сразу полностью обозреть. Именно, человек отмечает каким-либо способом поочередно по одной единице в обеих последовательностях (например, снабжая их штрихами) с исчерпанием одной последовательности выяснится, которая из них состоит из большего числа единиц.

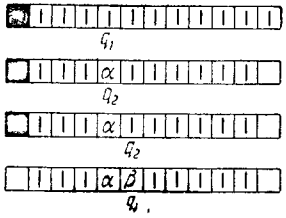


Рис. 17.

Машина же заменяет палочку первого числа символом α, потом заменяет палочку второго числа символом β, потом опять возвращается к палочкам первого числа и заменяет еще одну из них символом α, потом заменяет еще одну палочку второго числа символом β и т. д.

В первых четырех тактах на ленте складываются конфигурации, представленные на рис. 17, т. е. к концу

четвертого такта машина успела отметить уже по одной палочке каждого числа и теперь начинает сдвиг налево в поиске ближайшей, еще неотмеченной палочки левого числа. После еще нескольких тактов на ленте возникает конфигурация I рис. 18, т. е. первое число уже исчерпано, а второе еще нет. Поиски палочки слева не обнаружат таковой, а приведут к конфигурации II, и цикл сравнения уже осуществлен при участии одних лишь состояний q_1 и q_2. Следующий такт дает уже конфигурацию III.

Как видно из столбца состояния q_4 в схеме, представленной на рис. 8, теперь начинается сдвиг направо с заменой всех α пустыми знаками $\wedge$ (т. е. с вычеркиванием всех α) и заменой всех β палочками. После того как последняя справа β уже заменена палочкой, на ленте возникает конфигурация IV рис. 18, а вслед за тем конфигурация V. Таким образом, после цикла сравнения произошел цикл вычитания первого числа из второго, в результате которого меньшее число a стерто, а бо́льшее число b разбито на a, $b-a$; при этом обозревается последняя палочка

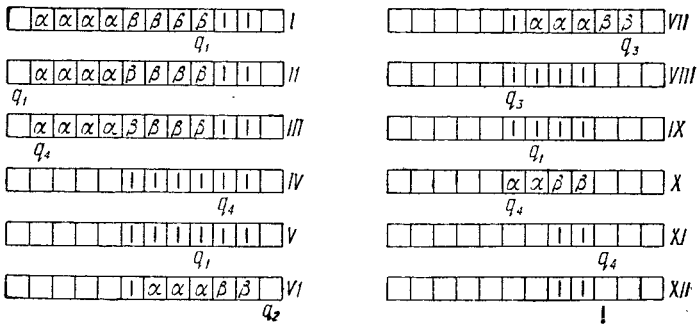


Рис. 18.

первого из этих чисел, а машина опять приходит в состояние q_1. Это означает, что первоначальная задача для чисел a, b сведена к такой же задаче для чисел a, $b-a$. Как раз на этом и основан, как мы уже знаем, алгоритм Евклида.

Дальше, разумеется, пойдет опять цикл сравнения. Теперь, однако, он заканчивается уже исчерпанием второго числа (справа), которое оказывается на этот раз меньшим. Последнее обнаруживается после того, как машина, заменив три палочки первого числа, не находит уже больше палочек во втором числе, т. е. возникает конфигурация VI.

Последующий такт порождает конфигурацию VII и с ним начинается цикл вычитания второго числа из первого, т. е. стирание всех β и замена всех α палочками. После замены последней слева α палочкой появляется на ленте конфигурация VIII, а затем и IX и тем самым цикл вычитания закончен и начинается следующий цикл сравнения и т. д. Этот процесс продолжается до тех пор, пока задача не будет сведена к случаю двух равных между собою чисел (в нашем примере это уже достигнуто). Тогда начинается последний цикл сравнения, который должен привести к результативному завершению процесса. Действительно, после того как получена конфигурация X, вычитание порождает конфигурацию XI и, наконец, результативную конфигурацию XII.

VI. Сочетание алгоритмов

При построении новых функциональных схем бывает удобным использовать ранее построенные схемы; это оказывается возможным в тех случаях, когда рассматривается алгоритм, являющийся в определенном смысле сочетанием алгоритмов, ранее уже изученных. Поясним это на примере. Пусть требуется построить функциональную схему алгоритма, перерабатывающего пару чисел a, b, заданных соответствующими наборами палочек, в их общий наибольший делитель, записанный в десятичной системе исчисления. Поскольку этот алгоритм может быть получен в результате *композиции*, т. е. последовательного применения двух ранее рассмотренных алгоритмов: сперва V, а потом II, его функциональная схема, изображенная на рис. 19, может быть также получена посредством надлежащего сочетания схем рис. 8 и рис. 12. Для этого сперва переименуем состояния схемы рис. 12 в p_0, p_1, p_2 с тем, чтобы отличить их от состояний схемы рис. 8, а в схеме рис. 8 знак остановки «!» заменим на p_2. Далее объединим эти измененные схемы в одну схему так, как это указано на рис. 19. (Клетки, оставленные нами незаполненными, соответствуют таким входным парам, которые в интересующем нас процессе не будут участвовать; в эти клетки можно вписать любые выходные тройки.) Теперь нетрудно проверить, что в соответствии со схемой рис. 19 сперва будет протекать процесс переработки двух заданных наборов палочек до тех пор, пока на ленте появится набор, изображающий общий наибольший делитель; однако, вместо остановки ! (см., например, конф. XII рис. 18) теперь

появится состояние p_2, и процесс будет продолжаться, обеспечивая дальнейшую переработку этого набора в десятичную запись числа.

Легко понять, что этот метод можно распространить на случай композиции любого конечного числа алгоритмов.

Отсюда, в частности, вытекает, что при рассмотрении численных алгоритмов можно считать, что натуральные

	q_1	q_2	q_3	q_4	p_0	p_1	p_2
0					$1\,p_2$	!	Π
1					$2\,p_2$	!	Π
2					$3\,p_2$	!	Π
3					$4\,p_2$	!	Π
4					$5\,p_2$	!	Π
5					$6\,p_2$	!	Π
6					$7\,p_2$	!	Π
7					$8\,p_2$	!	Π
8					$9\,p_2$	!	Π
9					$0\,\Pi$	!	Π
Λ	$\Pi\,q_4$	$\Pi\,q_3$	$\Pi\,q_1$	p_2	$1\,p_2$	!	Πp_1
!	$\alpha\,q_2$	$\beta\,q_1$	$\Pi\,q_1$	$\Pi\,q_1$	Π	$\Lambda\Pi p_0$	Π
α	Π	Π	! Π	$\Lambda\,\Pi$			
β	Π	Π	$\Lambda\,\Pi$	! Π			

Рис. 19.

числа заданы наборами палочек, ибо составив в таком предположении надлежащую схему $\mathfrak{A}$, можно от нее легко перейти к схеме $\mathfrak{B}$ применительно к десятичной системе счисления. Для этого достаточно построить схему $\mathfrak{B}$ по указанному выше методу путем композиции схем трех алгоритмов: алгоритма перевода из десятичной системы, алгоритма $\mathfrak{A}$ и алгоритма перевода в десятичную систему *).

Другим способом сочетания алгоритмов является многократное повторное применение одного и того же алгоритма до тех пор, пока не выполнено некоторое заранее указанное условие. Так, например, алгоритм перевода в десятич-

*) Здесь напрашивается сравнение с электронными вычислительными машинами, работающими в двоичной системе счисления: в них имеется устройство, переводящее исходные данные с десятичной системы в двоичную, и устройство, переводящее окончательный результат опять в десятичную систему.

ную систему сводится к повторному применению алгоритма, контролируемого перехода от n к $n+1$ до тех пор, пока не будут стерты все палочки. По функциональной схеме данного алгоритма (если такая уже имеется) и по предъявляемому условию можно строить схему для повторного алгоритма; однако этот метод сложнее, чем в случае композиции, и мы его не будем здесь подробно рассматривать.

Из разобранных примеров становится достаточно ясным, как составлять функциональные схемы и для других алгоритмов, в частности не численных. Укажем общий план составления функциональной схемы для алгоритма приведения слов (см. пример 4 § 3). Сперва составляются схемы $\mathfrak{A}_1$, $\mathfrak{A}_2$, $\mathfrak{A}_3$, $\mathfrak{A}_4$, которые реализуют ориентированные подстановки

$$b \longrightarrow acc$$
$$ca \longrightarrow accc$$
$$aa \longrightarrow \Lambda$$
$$cccc \longrightarrow \Lambda$$

соответственно. Так, например, машина со схемой $\mathfrak{A}_4$ перерабатывает любое слово в алфавите $\{a, b, c\}$, записанное на ленте, в слово, получаемое путем вычеркивания самой левой четверки рядом стоящих букв c; если же таких четверок нет, то слово оставляется неизменным. Далее строятся схемы $\tilde{\mathfrak{A}}_1$, $\tilde{\mathfrak{A}}_2$, $\tilde{\mathfrak{A}}_3$, $\tilde{\mathfrak{A}}_4$ для повторения алгоритмов $\mathfrak{A}_1$, $\mathfrak{A}_2$, $\mathfrak{A}_3$, $\mathfrak{A}_4$ соответственно. Так, например, машина со схемой $\tilde{\mathfrak{A}}_4$ вычеркивает одну четверку букв c, потом другую и т. д. до тех пор, пока не выполнено условие: получено слово, в котором нет ни одной четверки рядом записанных c. Наконец, искомая схема для алгоритма приведения представляет собою композицию схем $\tilde{\mathfrak{A}}_1$, $\tilde{\mathfrak{A}}_2$, $\tilde{\mathfrak{A}}_3$, $\tilde{\mathfrak{A}}_4$.

§ 9. ОСНОВНАЯ ГИПОТЕЗА ТЕОРИИ АЛГОРИТМОВ

Разбор предыдущих примеров создает впечатление о процессе, происходящем в тьюринговой машине, как о чрезвычайно замедленной киносъемке процесса вычисления, выполняемого человеком в соответствии с некоторым алгоритмом. Вместе с тем эти примеры подсказывают нам мысль задавать посредством функциональных схем тьюринговых машин и другие известные нам алгоритмы, которые обычно задаются иным способом, например в виде какого-либо словесного предписания или же в виде каких-либо специаль-

ных формул. Уже на данной стадии наших рассмотрений представляется весьма правдоподобным, что это действительно должно удаваться и в других случаях. Так ли оно на самом деле? Насколько общими являются понятия тьюринговой машины и тьюринговой функциональной схемы? Можно ли считать, что способ задания алгоритмов посредством функциональных схем является универсальным в том смысле, что всякий алгоритм может быть задан таким образом?

На эти вопросы современная теория алгоритмов предлагает ответ в виде следующей гипотезы:

Основная гипотеза теории алгоритмов. *Всякий алгоритм может быть задан посредством тьюринговой фукциональной схемы и реализован в соответствующей машине Тьюринга.*

Мы рассмотрим здесь два вопроса, возникающих в связи с формулировкой гипотезы:

1. В чем заключается значение этой гипотезы для теории алгоритмов?

2. В чем заключается обоснование гипотезы?

Прежде всего обратим внимание на следующую характерную особенность приведенной формулировки гипотезы. В этой формулировке речь идет, с одной стороны, о всяком алгоритме, т. е. об общем понятии алгоритма, которое, как уже подчеркивалось неоднократно, не является точным математическим понятием; с другой же стороны, в этой же формулировке речь идет о таком точном математическом понятии, как тьюрингова функциональная схема. Значение гипотезы как раз и заключается в том, что она уточняет общее, но расплывчатое понятие «всякого алгоритма» через более специальное, но уже совершенно точное математическое понятие тьюринговой функциональной схемы (и ее реализации в тьюринговой машине); таким образом, теория алгоритмов и объявляет в качестве объекта своего исследования всевозможные Тьюринговы функциональные схемы (Тьюринговы машины). Вместе с тем становятся уже осмысленными постановки таких вопросов, как вопрос о возможности или невозможности разрешающего алгоритма для задачи того или другого типа. Именно теперь это следует понимать как вопрос о существовании или несуществовании тьюринговой машины (функциональной схемы), обладающей нужным свойством.

Итак, сформулированная гипотеза оправдывает принятие основного определения современной теории алгоритмов, согласно которому расплывчатое понятие алгоритма отождест-

вляется с точным понятием функциональной схемы машины Тьюринга.

В чем же заключается обоснование этой столь важной гипотезы?

Заметим сперва, что не может быть и речи о том, чтобы доказать эту гипотезу подобно тому, как обычно доказываются в математике теоремы. Действительно, формулировка гипотезы не имеет характера теоремы, поскольку она представляет собою утверждение об общем понятии алгоритма, которое не является точным математическим понятием, а следовательно не может быть объектом строгих математических рассуждений.

Уверенность в справедливости гипотезы основана главным образом на опыте. Все известные алгоритмы, которые были придуманы в течение многих тысячелетий истории математики, могут быть заданы посредством тьюринговых функциональных схем. Правда, содержание гипотезы не обращено только в прошлое и не ограничивается констатацией того, что для всех известных алгоритмов оказалось возможным задание функциональными схемами. Содержание гипотезы имеет и вполне отчетливый характер прогноза на будущие времена: всякий раз, когда в будущем какое-либо предписание будет признано алгоритмом, то независимо от того, в какой форме и какими средствами это предписание будет первоначально выражено, его можно будет задавать также в виде функциональной схемы машины Тьюринга.

В этом смысле основную гипотезу можно сравнить с физическим законом, например с законом сохранения энергии, на основании которого также делаются прогнозы на будущее время. Именно громадный фактический опыт в прошлом признается достаточным основанием для подобных прогнозов.

Имеются и другие соображения, подтверждающие правильность основной гипотезы.

В предыдущем параграфе были указаны два приема построения из заданных исходных алгоритмов других более сложных алгоритмов: композиция алгоритмов и повторение алгоритма. Список таких приемов можно было бы еще продолжить. Однако все известные подобные приемы, а также все те, которые можно предвидеть при современном состоянии науки, оказываются таковыми, что, коль скоро для исходных алгоритмов возможно задание посредством функциональных схем, то оно возможно и для результирующих более сложных алгоритмов. В частности, для композиции

алгоритмов было показано, как по заданным функциональным схемам строится новая схема. Кроме того, напомним еще следующее обстоятельство, которое уже было упомянуто вскользь в § 6. Когда в науке возникла острая потребность в выработке точного понятия *алгоритм*, многие математики предприняли исследования с целью выявить некоторую стандартную форму задания алгоритмов, достаточно точную, для того чтобы она могла стать объектом математических исследований и достаточно общую, для того чтобы всем мыслимым алгоритмам можно было придавать такую форму. Кроме функциональных схем тьюринговых машин, были предложены и другие способы уточнения этого понятия. Так, например, А. А. Марков пришел к понятию *нормального алгоритма* (которое в этой книге было только бегло проиллюстрировано на алгоритме приведения для ассоциативного исчисления примера 4 § 3), а Гедель и Клини пришли к понятию *рекурсивного алгоритма* (*рекурсивной функции*) и т. д. Однако впоследствии все эти уточнения оказались равносильными. Этот факт нельзя считать случайным; он является дополнительным доводом в пользу сформулированной гипотезы.

Заметим еще в заключение, что внутри самой теории алгоритмов основная гипотеза не применяется, т. е. при доказательстве теорем этой теории никаких ссылок на основную гипотезу не делается. Таким образом, человек, не знающий этой гипотезы или же не признающий убедительность тех доводов, которые мы приводили в пользу ее правильности, не будет испытывать от этого каких-либо формальных затруднений в современной теории алгоритмов. Однако для такого человека то, что мы называем *теорией алгоритмов*, будет всего лишь теорией функциональных схем машин Тьюринга, т. е. по существу лишь теорией некоторых алгоритмов специального вида.

Автор этих строк вполне разделяет уверенность в справедливости основной гипотезы и вытекающей отсюда оценки современной теории алгоритмов, как общей теории, описывающей саму природу вещей, а не просто теории искусственно выделенного класса специальных «тьюринговых алгоритмов».

§ 10. УНИВЕРСАЛЬНАЯ МАШИНА ТЬЮРИНГА

До сих пор придерживались той точки зрения, что различные алгоритмы осуществляются в различных машинах Тьюринга, отличающихся своими функциональными схемами,

Однако можно построить *универсальную* тьюрингову машину, способную в известном смысле выполнять любой алгоритм, а значит способную выполнять работу любой тьюринговой машины.

Для того чтобы лучше уяснить себе, как это делается, представим себе следующий эксперимент. Пусть на ленту машины подана начальная информация $\mathfrak{A}$, и предположим, что некоторому человеку предложено указать, как будет перерабатывать машина эту информацию и во что она переработает ее окончательно. Если этот человек знаком с принципами работы тьюринговых машин, то достаточно ему сообщить, кроме этой начальной информации $\mathfrak{A}$, еще функциональную схему машины. Тогда человек, подражая работе машины и выписывая нужные конфигурации так, как мы это сделали при разборе алгоритма Евклида, сможет получить тот же результат, что и машина. Но это как раз и означает, что такой человек способен выполнять работу любой тьюринговой машины, если ему только задана ее функциональная схема. Сам же процесс подражания машине в соответствии с ее функциональной схемой может быть регламентирован в виде точного предписания, которое можно сообщить человеку, не имеющему ни малейшего понятия о машинах Тьюринга. Если человека, располагающего таким предписанием, которое естественно называть *алгоритмом подражания*, снабдить функциональной схемой какой-либо машины Тьюринга и, кроме того, некоторой начальной конфигурацией, изображенной на ленте, то он окажется способным в точности подражать работе соответствующей машины и в результате выдаст тот же результат. Подобный алгоритм подражания можно было бы задать хотя бы в виде следующей системы указаний:

Указание 1. Обозревай на ленте ячейку (единственную), под которой подписана буква.

Указание 2. Отыщи в таблице *) столбец, обозначенный такой же буквой, какая подписана под обозреваемой ячейкой.

Указание 3. В найденном столбце обозревай ту тройку букв, которая расположена на пересечении со строкой, обозначенной такой же буквой, какая вписана в обозреваемой ячейке.

---

*) То есть в функциональной схеме.

Указание 4. Замени букву из обозреваемой ячейки первой буквой из обозреваемой тройки.

Указание 5. Если в обозреваемой тройке второй буквой является *I*, то остановись; процесс окончен.

Указание 6. Если в обозреваемой тройке второй буквой является *H*, то замени букву, подписанную под обозреваемой ячейкой, третьей буквой из обозреваемой тройки.

Указание 7. Если в обозреваемой тройке второй буквой является *Л*, то сотри букву, подписанную под обозреваемой ячейкой, и левее ее запиши третью букву из обозреваемой тройки.

Указание 8. Если в обозреваемой тройке второй буквой является *П*, то сотри букву, подписанную под обозреваемой ячейкой, и правее ее запиши третью букву из обозреваемой тройки.

Указание 9. Переходи к указанию 1.

Но оказывается, что вместо человека, действующего в соответствии с алгоритмом, можно поставить некоторую тьюрингову машину; это и будет универсальная тьюрингова машина, способная подражать работе любой другой тьюринговой машины. Иными словами, это означает, что алгоритм подражания, словесно описанный нами выше посредством системы девяти указаний, может быть надлежащим образом задан в виде некоторой Тьюринговой функциональной схемы (универсальной схемы). Полное и строгое доказательство этого факта, представляющего собою еще одно подтверждение основной гипотезы теории алгоритмов, слишком громоздко в своих деталях для того, чтобы его можно было воспроизвести в этой небольшой книге. Мы ограничимся некоторыми общими указаниями, которых будет, по-видимому, достаточно для понимания существа дела.

Заметим вначале, что в описанном нами алгоритме подражания в качестве исходных данных (исходной информации) фигурируют функциональная схема подражаемой машины и соответствующая начальная конфигурация. Эта исходная информация перерабатывается алгоритмом в окончательную конфигурацию, изображающую результат, выдаваемый подражаемой машиной. Универсальная машина должна делать то же самое. Однако здесь приходится учитывать следующие два обстоятельства:

1. Непосредственная подача функциональной схемы подражаемой машины и соответствующей конфигурации на ленту универсальной машины в качестве исходной инфор-

мации невозможна. Действительно, в универсальной машине, как и во всякой тьюринговой машине, информация изображена буквами, расположенными на ленте одномерно, т. е. в одной строке, образуя одно или несколько слов во внешнем алфавите машины. В то же время функциональные схемы мы до сих пор задавали посредством «двумерных» таблиц, в которых буквы располагаются несколькими строками; аналогично обстоит дело с конфигурациями, в которых буквы состояний записаны под буквами внешнего алфавита (под лентой).

2. Универсальная машина (как и всякая тьюрингова машина) может располагать лишь фиксированным конечным внешним алфавитом. Между тем, она должна быть приспособлена к приему в качестве исходной информации всевозможных схем и конфигураций, в которых могут встречаться буквы из разнообразных алфавитов со сколь угодно большим числом различных букв.

Поэтому мы должны в первую очередь позаботиться о выработке надлежащего способа задания функциональных схем и конфигураций, который соответствовал бы указанным выше особенностям любой отдельно взятой тьюринговой машины, а именно одномерности информации и конечности алфавита. К описанию такого способа мы сейчас и переходим:

1. Вместо того, чтобы изобразить схему в виде двумерной таблицы, насчитывающей k строк и m столбцов, распишем одну за другой mk пятерок букв из этой таблицы так, что первый символ пятерки указывает столбец таблицы, второй — строчку таблицы, а последующие три — символы той тройки, которая располагается в таблице на пересечении указанной строки с указанным столбцом.

Так, например, вместо схемы рис. 6 возникает одномерная строка символов

$$q_1 \mid \alpha H q_2 q_2 \mid \beta H q_1 q_3 \parallel \Pi q_1 \ldots \qquad (\Omega)$$

Очевидно, по этой строке можно при желании одно начным образом восстановить первоначальную таблицу. Поступая аналогично, можно при рассмотрении конфигураций условиться в том, чтобы букву состояний писать не под обозреваемой буквой, а непосредственно левее ее. В таком случае конфигурация IV на рис. 18 перейдет в строку

$$\mid\mid\mid\mid q_4 \mid\mid .$$

Очевидно, по такому одномерному изображению конфигурации можно при желании однозначным образом восстановить ее первоначальный вид.

2. Для характеристики функциональной схемы и конфигураций вовсе не существенны специфические начертания букв внешнего алфавита и алфавита состояний, которые в ней фигурируют. Так, например, если всюду в таблице рис. 6 или же в соответствующей ей строке заменить букву β буквой b, то от этого ничего в наших рассмотрениях не изменится. Важно лишь то, чтобы различные объекты были заданы различными символами и чтобы можно было различить буквы состояний от букв внешнего алфавита.

Конечно, можно было выбрать другие буквы, отличные от *Л*, *П*, *Н*, и для обозначения сдвигов (влево, вправо, отсутствие сдвига), однако при этом должно быть четко оговорено, какой именно буквой какой сдвиг обозначается. Здесь сказывается тот факт, что каждая из трех букв обозначает вполне определенное действие, которое не заменимо другим.

Учитывая эти обстоятельства, заменим в строке $\mathfrak{Q}$ каждую отдельную букву некоторой последовательностью из единиц и нулей (*кодовой группой*) так, чтобы различные буквы заменялись различными кодовыми группами, но одна и та же буква заменялась бы всюду, где бы она ни встречалась, одной и той же кодовой группой. В результате такой замены, например, строка $\mathfrak{Q}$ перейдет в некоторую строку $\mathfrak{Q}'$. Для того чтобы по $\mathfrak{Q}'$ можно было восстановить $\mathfrak{Q}$, способ кодирования (отнесения кодовых групп буквам) должен удовлетворять следующим условиям:

1) чтобы строку $\mathfrak{Q}'$ можно было бы однозначным образом разбить на отдельные кодовые группы;

2) чтобы можно было распознавать, какие кодовые группы отнесены каждой из букв *Л*, *П*, *Н* в отдельности, и чтобы можно было различить кодовые группы, отнесенные буквам состояний от тех, которые отнесены буквам внешнего алфавита.

Эти два условия будут наверняка соблюдены при следующем способе кодирования.

1. В качестве кодовых групп берутся $3+k+m$ различных слов вида

$$100\ldots01$$

(между единицами — сплошь нули).

Тогда разбиение строки $\mathfrak{Q}'$ на кодовые группы производится однозначным образом и легко, путем выделения последовательностей нулей, заключенных между двумя единицами.

2. Сопоставление кодовых групп исходным буквам осуществляется согласно следующей таблице кодирования:

	Буква	Кодовая группа	
	$Л$	101	
	$Н$	1001	
	$П$	10001	
Внешний алфавит	S_1	100001 (4 нуля)	Четное число нулей, большее чем 2
	S_2	10000001 (6 нулей)	
	. . . . .		
	S_k	10....01 ($2(k+1)$ нулей)	
Алфавит состояний	q_1	1000001 (5 нулей)	Нечетное число нулей, большее чем 5
	q_2	100000001 (7 нулей)	
	. . . . .		
	q_m	10.....01 ($2(m+1)+1$ нулей)	

При такой системе кодирования в нашем случае $\mathfrak{Q}'$ выглядит так:

| 00003 || 0030 || 000000 || 00 || 0003000 || 0000000 || 0000 || 00000000 || 00 || 00000 | ...

Подобную строчку из единиц и нулей, составленную для функциональной схемы или для конфигураций, будем называть *шифром функциональной схемы* или *шифром конфигурации* соответственно. По заданному шифру сама схема или конфигурация в своем первоначальном виде легко восстанавливается; поэтому задание схемы или конфигурации можно всегда осуществить посредством их шифров. Разумеется, вместо единицы и нулей можно было бы брать любые другие два знака, например a, b.

Теперь уже нетрудно сообразить, как изменить формулировки указаний 1—9 из первоначального описания алгоритма подражания для того, чтобы получить алгоритм, который перерабатывает шифр схемы подражаемой машины и шифр начальной конфигурации в шифр результирующей конфигурации. Ограничимся только отдельными иллюстрациями:

Указание 1. Обозревай в шифре конфигурации кодовую группу (единственную), расположенную непосредственно правее кодовой группы с нечетным числом нулей.

Указание 2 и 3. Отыщи в шифре схемы пару соседних кодовых групп, одинаковую с парой кодовых групп в шифре конфигурации, в которой вторая группа является обозреваемой.

Указание 6. Если в обозреваемой тройке кодовых групп из шифра схемы второй является группа 1001, то в шифре конфигурации замени кодовую группу с нечетным числом нулей третьей кодовой группой из этой обозреваемой тройки.

Дальнейшее рассмотрение этого алгоритма позволяет сводить каждую операцию над кодовыми группами к цепи стандартных операций, осуществимых в тьюринговой машине (замена одного знака другим, сдвиг на один шаг и т. п.). При этом, кроме знаков 1 и 0, из которых построены шифры, будут участвовать и другие буквы, например буква, отделяющая один шифр от другого, буквы, играющие роль временных пометок при просмотре единиц и нулей (сравнить с алгоритмом Евклида) и другие.

В результате такой детализации алгоритм подражания оказывается в конечном счете описанным некоторой тьюринговой функциональной схемой. Это и есть схема универсальной машины. Если какая-либо машина A решает некоторую задачу, то и универсальная машина способна решить эту задачу при условии, что кроме шифра исходных данных этой задачи на ее ленту будет подан шифр схемы машины A.

В связи с существованием универсальной тьюринговой машины всевозможные функциональные схемы (или их шифры) можно толковать двояко:

1) схема описывает логический блок специальной машины Тьюринга, реализующей соответствующий алгоритм (это и есть точка зрения, которая проводилась первоначально);

2) схема описывает программу, подаваемую на ленту универсальной машины для реализации соответствующего алгоритма.

Заметим в заключение, что современные вычислительные электронные машины и строятся как раз как универсальные машины, в запоминающее устройство которых наряду с исходными данными поставленной задачи вводится также и программа ее решения.

Разделение памяти на внешнюю и внутреннюю характерно и для вычислительных машин, причем внешняя память представлена чаще всего магнитными барабанами и лентами, на которых производится магнитная запись информации, наподобие обычной звукозаписи. Однако в отличие от машины Тьюринга, в которой внешняя память бесконечна (лента бесконечна), в любой реальной вычислительной машине внешняя память (магнитная лента или барабан) конечна.

Разумеется, это принципиальное различие между реальной вычислительной машиной и машиной Тьюринга, представляющей собой некоторую абстрактную, идеализированную машину, является неустранимым. Вместе с тем важно отметить, что в реальной вычислительной машине внешнюю память можно неограниченно увеличивать без изменения конструкции машины; для этого достаточно к уже имеющемуся «куску» магнитной ленты, «приклеить» дополнительный «кусок».

§ 11. АЛГОРИТМИЧЕСКИ НЕРАЗРЕШИМЫЕ ПРОБЛЕМЫ

Переход от расплывчатого понятия алгоритма к точному понятию машины Тьюринга, могущей быть заданной своим шифром, позволяет уточнить и вопрос об алгоритмической (или машинной) разрешимости того или иного круга задач. Именно теперь этот вопрос следует понимать так: существует ли машина Тьюринга, решающая данный класс задач, или же такой машины не существует ? (что значит «машина Тьюринга решает некоторый класс задач» — см. § 7).

На этот вопрос теория алгоритмов в ряде случаев дает отрицательный ответ. Один из первых результатов такого типа, установленный американским математиком Чёрчем в 1936 году, относится и к проблеме распознавания выводимости в математической логике (см. § 6).

Теорема Чёрча. *Проблема распознавания выводимости алгоритмически неразрешима.*

Тем самым не только выясняется причина безуспешности всех прошлых попыток создания соответствующих алгоритмов, но и обнаруживается полная бессмысленность таких попыток.

Доказательствам невозможности, проводимым в теории алгоритмов, присуща та же математическая строгость, что и доказательствам невозможности, проводимым в других

областях математики (например, невозможность трисекции угла с помощью циркуля и линейки, или невозможность определения общей меры для стороны квадрата и его диагонали). Эскиз такого доказательства мы приведем сейчас для *проблемы распознавания самоприменимости.*

Пусть на ленте машины Тьюринга изображен ее же собственный шифр (т. е. шифр функциональной схемы машины), записанный в алфавите машины. Возможны два случая: 1) машина применима к своему шифру, т. е. она перерабатывает этот шифр и после конечного числа тактов останавливается, подавая сигнал об остановке; 2) машина не применима к своему шифру, т. е. сигнал об остановке никогда не наступает. В связи с этим сами машины (шифры) разбиваются на два класса: класс *самоприменимых* и класс *несамоприменимых* тьюринговых машин (шифров). Возникает следующая массовая проблема.

Проблема распознавания самоприменимости. *По любому заданному шифру установить, к какому классу относится машина, зашифрованная им: к классу самоприменимых или несамоприменимых?*

Это — типичная задача на построение алгоритма, ибо для ее решения надо найти общий метод (алгоритм или машину), позволяющий автоматически решать по любому данному шифру, является ли он самоприменимым, или нет.

Теорема. *Проблема распознавания самоприменимости алгоритмически неразрешима.*

Доказательство. Предположим от противного, что такая машина A существует. Тогда в A всякий самоприменимый шифр перерабатывается в какой-то символ σ (имеющий смысл утвердительного ответа на поставленный вопрос о самоприменимости), а всякий несамоприменимый шифр в другой символ τ (имеющий смысл отрицательного ответа на поставленный вопрос). В таком случае можно было бы построить и такую машину B, которая по-прежнему перерабатывает несамоприменимые шифры в τ, в то время как к самоприменимым шифрам B уже не применима. Этого можно было бы добиться путем такого изменения схемы машины B, чтобы после появления символа σ вместо сигнала об остановке машина стала бы неограниченно повторять этот же символ.

Итак, B применима ко всякому несамоприменимому шифру (вырабатывая при этом символ τ) и не применима к самоприменимым шифрам. Однако это приводит к про-

тиворечию. Действительно: 1) пусть эта машина B самоприменима, тогда она применима к своему шифру B и перерабатывает его в символ τ; но ведь появление этого символа как раз и должно означать, что B несамоприменима; 2) пусть B несамоприменима, тогда она не применима к B, что должно означать как раз, что B (B') самоприменима. Полученное противоречие доказывает теорему.

Первые результаты об алгоритмической неразрешимости были установлены для проблем, возникающих в самой математической логике (проблема выводимости) и в теории алгоритмов (например, проблема самоприменимости). Однако позднее выяснилось, что подобные явления имеют место и для некоторых казалось бы более узких проблем, возникающих в самых разнообразных специальных разделах математики.

В первую очередь здесь следует указать на ряд алгебраических проблем, приводящих к различным вариантам проблемы слов, которые исследовались советскими математиками.

Проблема эквивалентности слов для ассоциативных исчислений (см. § 3) была сформулирована еще в 1914 году норвежским математиком Туэ; им же был предложен алгоритм для распознавания эквивалентности слов в некоторых ассоциативных исчислениях специального вида. С тех пор предпринимались многие попытки построить такой общий алгоритм, который для любого ассоциативного исчисления и для любой пары слов в нем позволил бы установить, эквивалентны ли эти слова или нет. В 1946 и 1947 годах советский математик Андрей Андреевич Марков и американский математик Эмиль Пост, независимо один от другого, построили конкретные примеры ассоциативных исчислений, для каждого из которых проблема эквивалентности слов алгоритмически неразрешима. Тем более не существует алгоритма для распознавания эквивалентности слов в любом исчислении. Впоследствии на основе этого результата А. А. Марков и его ученики установили невозможность распознающих алгоритмов для широкого класса свойств ассоциативных исчислений.

Большое впечатление произвел в математическом мире результат Петра Сергеевича Новикова об алгоритмической неразрешимости проблемы тождества теории гру п, опубликованный в 1955 году*). Формально эта проблема представляет собою частный случай проблемы эквивалентности слов

*) За эту работу П. С. Новикову в 1957 г. присуждена Ленинская премия.

в ассоциативном исчислении*). Именно рассматриваются лишь такие ассоциативные исчисления, в которых для каждой буквы a алфавита в списке допустимых подстановок исчисления имелась бы подстановка вида

$$a\alpha — \Lambda,$$

где α — какая-либо буква того же алфавита (возможно, совпадающая с a).

Содержательный смысл этого требования становится понятным, если, по аналогии с примером 4 из § 3, истолковать слова в любом ассоциативном исчислении как некоторые сложные преобразования, получаемые путем умножения элементарных преобразований, задаваемых соответствующими буквами, образующими это слово. В таком случае пустое слово Λ задает тождественное преобразование, ничего не изменяющее (сравни с § 3), а наличие допустимых подстановок вида $a\alpha — \Lambda$ означает, что для каждого элементарного преобразования (задаваемого буквой a) существует элементарное преобразование (задаваемое буквой α) такое, что последовательное их осуществление дает тождественное преобразование. Не вдаваясь в дальнейшие подробности, отметим лишь, что рассмотрение таких множеств преобразований, называемых группами преобразований, представляет исключительно большой теоретический и практический интерес, а само понятие группы является одним из самых основных понятий современной математики.

Теперь нам нужно уяснить себе, что весьма важный результат Маркова — Поста, отмеченный выше, сам по себе не позволяет делать никаких выводов по существу проблемы тождества теории групп. Дело в том, что те индивидуальные ассоциативные исчисления, для которых А. А. Марков и Э. Пост установили алгоритмическую неразрешимость проблемы эквивалентности, как раз не удовлетворяют требованию, отмеченному выше, которое является существенным при постановке проблемы тождества теории групп; поэтому возможность алгоритма для этой последней проблемы не

*) Это означает, что если бы существовал алгоритм, выясняющий тождественность слов в ассоциативном исчислении, то этот же алгоритм устанавливал бы тождественность слов в группе. Однако из алгоритмической неразрешимости проблемы тождества слов в ассоциативном исчислении вовсе не следует алгоритмическая неразрешимость соответствующей проблемы в теории групп. (*Прим. ред.*)

исключается результатами Маркова — Поста. Надежда на создание этого алгоритма еще не была полностью утрачена, и поиски его еще продолжались, когда стал известным результат П. С. Новикова, согласно которому такого алгоритма не существует. П. С. Новиков построил индивидуальный пример ассоциативного исчисления, удовлетворяющего указанному требованию, для которого невозможен алгоритм, распознающий эквивалентность; тем более невозможен единый алгоритм для всех рассматриваемых групп.

Примеры, построенные А. А. Марковым и П. С. Новиковым для опровержения алгоритмической разрешимости исследуемых проблем, оказались весьма громоздкими и насчитывали сотни допустимых подстановок. Была поставлена задача построения по возможности более простых подобных примеров. Эта задача недавно была блестяще решена молодым ленинградским математиком Г. С. Цейтиным; для приведенного в § 3 исчисления, построенного Цейтиным и насчитывающего всего лишь семь допустимых подстановок, проблема эквивалентности слов также алгоритмически неразрешима.

Обнаружение алгоритмически неразрешенных проблем создало в науке такую ситуацию, когда математик, стремящийся к построению желаемого алгоритма, должен считаться с тем, что такого алгоритма может и не существовать. Поэтому параллельно с усилиями, направленными на поиски желаемого алгоритма, приходится прилагать усилия на доказательство невозможности такого алгоритма. В зависимости от того, на каком из этих направлений будет достигнут успех и выяснится окончательно картина, либо будет найден разрешающий алгоритм, либо будет установлена алгоритмическая неразрешимость проблемы.

В § 1 нами была сформулирована проблема Гильберта о диофантовых уравнениях. В течение полувека велись односторонние, причем безуспешные исследования с целью построить желаемый алгоритм. В последнее время уже проводятся исследования и в противоположном направлении и хотя окончательного результата пока еще нет, но в свете частичных результатов, которые уже получены, кажется уже правдоподобным, что проблема Гильберта также является алгоритмически неразрешимой.

ЗАКЛЮЧИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

Сделаем в заключение несколько общих замечаний.

1. Во-первых, теоремы об алгоритмической неразрешимости того или другого класса задач не дают повода для впадания в агностицизм. Действительно, каждая такая теорема касается целого класса задач и устанавливает неразрешимость всех задач этого класса единым эффективным методом — алгоритмом.

Это вовсе не означает, что среди единичных задач объединяемых этим классом, имеются такие, которые неразрешимы. Например, доказанную выше теорему не следует понимать так, что существует такой шифр, о котором в принципе невозможно установить, является ли он самоприменимым или нет.

Это означает лишь, что рассматриваемый тип задач является столь общим и широким, что единого алгоритма для решения всех задач данного типа не существует. В этом случае целью математических исследований является последовательное создание все более и более широких алгоритмов, позволяющих сводить к автоматическому вычислению все более и более обширные подклассы задач данного типа.

2. Во-вторых, теоремы об алгоритмической неразрешимости показывают, что математика не сводится к построению алгоритмов, что процесс познания в математике не может быть до конца автоматизирован. Уже в отдельных, сравнительно узких областях математики (как теория групп с конечным числом образующих и т. д.) возникают массовые проблемы, решить которые не способен никакой автомат (т. е. никакая машина Тьюринга с конечным числом состояний и конечной памятью). Тем более абсурдны утверждения будто машина полностью может заменить творческий труд ученого.

3. Вместе с тем приходится признать, что область применения алгоритмических процессов весьма широка и к ней относятся не только вычислительные процессы, встречающиеся в математике. Более того, для многих процессов, которые обычно принято считать очень трудными и сложными, можно теоретически построить алгоритмы, являющиеся по своей идее достаточно простыми; практические же трудности, встречаемые при реализации этих процессов, связаны с тем, что указанные алгоритмы являются очень длинными и требуют совершения огромного числа операций (хотя эти операции сами по себе простые). Это замечание относится, в частности, к процессам игры (и в частности — игры в шахматы), где успех во многом зависит от умения обозреть б о́ л ь ш е е число вариантов для выбора оптимального варианта.

С созданием быстродействующих вычислительных машин мы значительно расширили число практически осуществимых алгоритмов.

4. Наконец, обратим еще раз внимание на то, что каждая физически осуществимая вычислительная машина может быть рассматриваема лишь как некоторая приближенная модель машины Тьюринга. Именно, в реальных машинах объем внешней памяти ограничен, в то время как в схем машины Тьюринга фигурирует бесконечная лента. Разумеется техническое осуществление неограниченной памяти невозможно, но значительное увеличение объема памяти в машинах по сравнению с уже достигнутым уровнем не только желательно, но и вполне возможно. Именно в этом направлении наращивания объема внешней памяти и скорости вычисления можно ожидать дальнейших больших успехов в развитии вычислительных автоматов.

Трахтенброт Борис Авраамович.
Алгоритмы и машинное решение задач.

Редактор *М. М. Горячая.*
Техн. редактор *Е. А. Ермакова.*
Корректор *И. С. Цветкова.*

Сдано в набор 4/IX 1957 г. Подписано к печати 27/XI 1957 г. Бумага 84×108/32. Физ. печ. л. 3,0. Условн. печ. л. 4,92. Уч.-изд. л. 5,31. Тираж 25 000 экз. Т-10379.
Цена книги 1 р. 60 к. Заказ 2433

Государственное издательство
технико-теоретической литературы
Москва, В-71, Б. Калужская, 15.

Типография № 2 им. Евг. Соколовой
УПП Ленсовнархоза.
Ленинград, Измайловский пр., 29

Цена 1 р. 60 к.

ГОСУДАРСТВЕННОЕ ИЗДАТЕЛЬСТВО ТЕХНИКО-ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ

ПОПУЛЯРНЫЕ ЛЕКЦИИ ПО МАТЕМАТИКЕ

Вып. 1. **А. И. Маркушевич.** Возвратные последовательности.
Вып. 2. **И. П. Натансон.** Простейшие задачи на максимум и минимум.
Вып. 3. **И. С. Соминский.** Метод математической индукции.
Вып. 4. **А. И. Маркушевич.** Замечательные кривые
Вып. 5 **П. П. Коровкин.** Неравенства.
Вып. 6. **Н. Н. Воробьев.** Числа Фибоначчи.
Вып. 7. **А. Г. Курош.** Алгебраические уравнения произвольных степеней.
Вып. 8. **А. О. Гельфонд.** Решение уравнений в целых числах.
Вып. 9. **А. И. Маркушевич.** Площади и логарифмы.
Вып. 10. **А. С. Смогоржевский.** Метод координат
Вып. 11. **Я. С. Дубнов.** Ошибки в геометрических доказательствах.
Вып. 12. **И. П. Натансон.** Суммирование бесконечно малых величин.
Вып. 13. **А. И. Маркушевич.** Комплексные числа и конформные отображения.
Вып. 14. **А. И. Фетисов.** О доказательствах в геометрии.
Вып. 15. **И. Р. Шафаревич.** О решении уравнений высших степеней.
Вып. 16. **В. Г. Шерватов.** Гиперболические функции.
Вып. 17. **В. Г. Болтянский.** Что такое дифференцирование?
Вып. 18. **Г. М. Миракьян.** Прямой круговой цилиндр.
Вып. 19. **Л. А. Люстерник.** Кратчайшие линии.
Вып. 20. **А. М. Лопшиц.** Вычисление площадей ориентированных фигур.
Вып. 21. **Л. И. Головина и И. М. Яглом.** Индукция в геометрии.
Вып. 22. **В. Г. Болтянский.** Равновеликие и равносоставленные фигуры.
Вып. 23. **А. С. Смогоржевский.** О геометрии Лобачевского.
Вып. 24. **Б. И. Аргунов и Л. А. Скорняков.** Конфигурационные теоремы.
Вып. 25. **А. С. Смогоржевский.** Линейка в геометрических построениях.
Вып. 26. **Б. А. Трахтенброт.** Алгоритмы и машинное решение задач.